Владимирский государственный университет

С. А. САМОЙЛОВ В. С. САМОЙЛОВ

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ ДЛЯ БЕСПРОВОДНЫХ СИСТЕМ СВЯЗИ

Учебно-практическое пособие

Владимир 2023

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

С. А. САМОЙЛОВ В. С. САМОЙЛОВ

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ ДЛЯ БЕСПРОВОДНЫХ СИСТЕМ СВЯЗИ

Учебно-практическое пособие

Владимир 2023

УДК 621.396
ББК 32.844
С17

Рецензенты:
Доктор технических наук, профессор
зав. кафедрой конструирования и технологии радиоэлектронных средств
Арзамасского политехнического института (филиала) Нижегородского
государственного технического университета им. Р. Е. Алексеева
Н. П. Ямпурин

Доктор технических наук, профессор
лауреат премии Совета Министров СССР
главный инженер проектов
Отдела проектного менеджмента АО «Гипрогазцентр» (г. Нижний Новгород)
С. В. Ларцов

Издается по решению редакционно-издательского совета ВлГУ

Самойлов, С. А. Программирование микроконтроллеров
С17 для беспроводных систем связи : учеб.-практ. пособие / С. А. Самойлов, В. С. Самойлов ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2023. – 164 с.
ISBN 978-5-9984-1547-0

Рассмотрены современные микроконтроллеры, их архитектура и устройства ввода/вывода. Описаны основные функции, операторы и методы программирования микроконтроллеров на базе платформы Arduino. Приведены основы программирования беспроводных модулей связи. Представлены методы и примеры помехоустойчивого кодирования информации и псевдослучайной перестройки частоты.

Предназначено для студентов и магистрантов очной и заочной форм обучения направлений подготовки 11.03.01, 11.04.01 «Радиотехника» и 11.03.02 «Инфокоммуникационные технологии и системы связи», будет полезно специалистам в области проектирования радиотехнических устройств и систем.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Табл. 5. Ил. 63. Библиогр.: 40 назв.

УДК 621.396
ББК 32.844

ISBN 978-5-9984-1547-0

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Появление в середине прошлого века программируемых электронных вычислительных систем привело к бурному развитию вычислительной техники. Изобретение микропроцессора, а также интегрированных микросхем оперативной памяти обусловило зарождение высокоуровневых языков программирования. Разработка микроконтроллеров, где в одном корпусе сочетаются микропроцессор, память и устройства ввода/вывода, позволила решать множество задач по управлению электротехническими устройствами, сбору и анализу информации, а также организации беспроводных систем связи. Факторы доступности, низкой стоимости и относительной простоты программирования обусловили широкое применение микроконтроллеров в современной науке и технике.

Важной особенностью микроконтроллеров является возможность его перепрограммирования, что позволяет значительно уменьшить время разработки электронного устройства и снизить негативное влияние от возможных ошибок при разработке. Кроме того, перепрограммирование микроконтроллеров позволяет изменить алгоритм работы электронного устройства без изменения его аппаратной части. Таким образом, можно создавать электронные устройства с неизменными аппаратными средствами, но различными алгоритмами функционирования в зависимости от решаемой задачи.

Наряду с традиционными разделами курса по изучению микроконтроллеров, такими как архитектура микроконтроллеров, устройства ввода/вывода, интерфейсы, средства программирования, операторы и функции программирования, обработка аппаратных и программных прерываний, в книгу включены разделы по организации беспроводных систем связи с помощью микроконтроллеров. В связи с бурным развитием беспроводных систем связи в пособии приведены базовые основы программирования трансиверов, применение помехоустойчивых кодеков и организации системы псевдослучайной перестройки радиочастоты. Многообразие существующих микроконтрол-

леров, их интерфейсов, устройств ввода/вывода, а также средств и языков программирования заставили авторов ограничиться рассмотрением основных и практически значимых устройств и разделов, отсылая читателей по вспомогательным вопросам к известной литературе.

В первой главе изложена краткая история развития микропроцессорной техники и разработки микроконтроллеров. Приведены наиболее популярные на сегодняшний день микроконтроллеры различных фирм-производителей, описание устройств ввода/вывода; рассмотрена архитектура микроконтроллеров; показаны принципы функционирования интегрированных аналого-цифровых и цифроаналоговых преобразователей. Особое внимание уделено интерфейсам микроконтроллеров, например UART/USART, SPI, JTAG. Кроме того, приведены электрические схемы подключения к различным интерфейсам.

Во второй главе изложены основы программирования микроконтроллеров на базе платформы Arduino как наиболее популярной на сегодняшний день и позволяющей достаточно быстро овладеть общими навыками программирования.

Приведено описание платформы Arduino, цифровых и аналоговых входов и выходов; дано представление об оперативной памяти и широтно-импульсной модуляции; подробно указаны основные функции, типы данных и операторы программирования на языке C++; раскрыты принципы организации программных прерываний по таймеру и приведен пример программы.

В третьей главе показаны основы организации беспроводных систем связи; приведен обзор современных трансиверов и указаны наиболее важные характеристики беспроводных модулей передачи данных; описаны устройство и регистры программирования беспроводного модуля NRF24L01 и его подключение к платформе Arduino; изложены принципы программирования приемника и передатчика для беспроводной системы связи, а также понятие помехоустойчивого кодирования на примере кодов Хемминга и Рида – Соломона. Рассмотрена система связи с псевдослучайной перестройкой радиочастоты.

В приложениях приведены полные тексты программ для организации беспроводных систем связи с помехоустойчивым кодированием по коду Хемминга, с корректирующим помехоустойчивым кодом Ри-

да – Соломона, а также система связи с псевдослучайной перестройкой радиочастоты.

Пособие обобщает материал лекционных курсов, читавшихся авторами более 20 лет во Владимирском государственном университете имени Александра Григорьевича и Николая Григорьевича Столетовых, а также результаты научно-исследовательских работ, выполнявшихся авторами по проектированию мощных генераторов сигналов, усилителей мощности и устройств помехоустойчивого кодирования – кодеров и декодеров. Разделы, посвященные системам помехоустойчивого кодирования, содержат материал, который сконцентрирован в учебной литературе, излагается впервые и значительная его часть получена самостоятельно авторами при проведении исследовательских и опытно-конструкторских работ.

Авторы выражают глубокую благодарность доктору технических наук, профессору, заведующему кафедрой «Конструирование и технология радиоэлектронных средств» Арзамасского политехнического института (филиала) Нижегородского государственного технического университета им. Р. Е. Алексеева Н. П. Ямпурину и главному инженеру проектов Отдела проектного менеджмента АО «Гипрогазцентр» (г. Нижний Новгород), доктору технических наук, профессору, лауреату премии Совета Министров СССР С. В. Ларцову за обсуждение материала и ценные советы по улучшению содержания книги.

Глава 1. МИКРОКОНТРОЛЛЕРЫ

1.1. История развития микроконтроллеров

Микроконтроллеры являются неотъемлемой частью быта современного человека. Они применяются во многих изделиях от детских игрушек до АСУТП (автоматизированная система управления технологическим процессом). Благодаря использованию микроконтроллеров инженерам удалось достигнуть большой скорости изготовления и высокого качества продукции практически во всех сферах производства.

Чтобы разобраться с причинами появления и развития микропроцессорной техники, необходимо обратить внимание на характеристики и особенности первых компьютеров. На рис. 1.1 показан ENIAC – первый компьютер (1946 г.). Он занимал целое помещение, или 85 м^3 объема в пространстве, и весил 30 т, обладая большим тепловыделением и энергопотреблением, отмечались постоянные неполадки из-за разъемов электронных ламп. Окислы приводили к исчезновению контактов, лампы теряли связь с платой и требовали постоянного обслуживания [1].

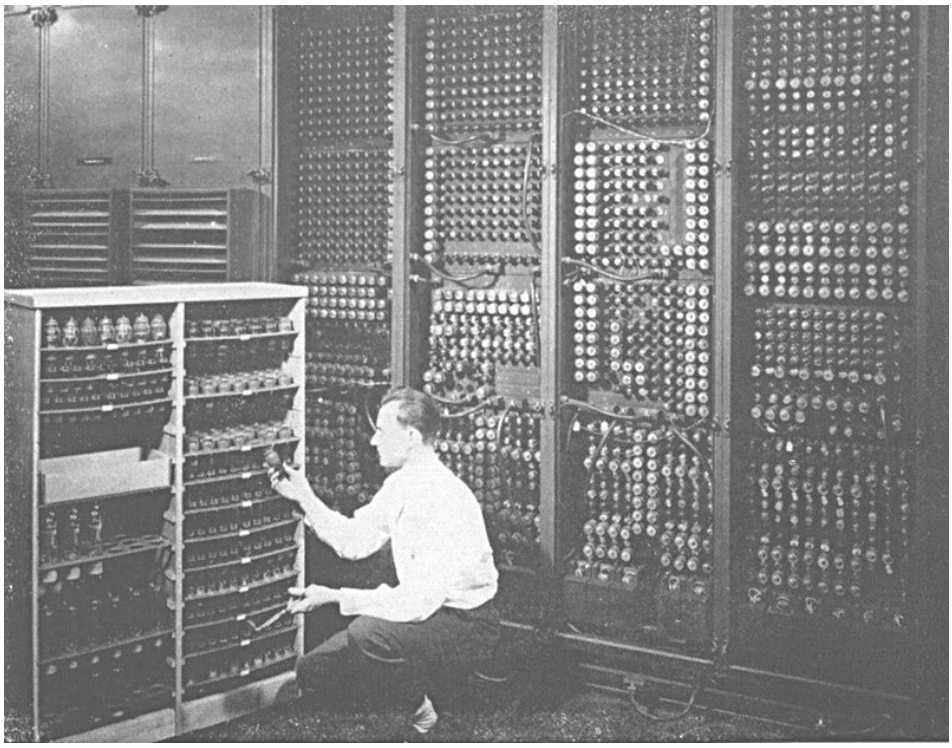

Рис. 1.1. ENIAC – первый компьютер

Развивались полупроводниковые технологии: в 1907 году были проведены исследования по детекторам и электролюминесценции полупроводников, в 1940-е годы – по диодам и транзисторам. Это все привело к появлению интегральных технологий. Роберт Нойс в 1959 году изобрел интегральную микросхему (дальше ИМС или МС).

Компьютерная техника развивалась и к концу 60-х гг. прошлого века в мире было порядка 30 тыс. компьютеров, в их числе как универсальные ЭВМ, так и мини-компьютеры. Мини-компьютеры того времени были размерами со шкаф. В 1969 году был изобретен прообраз Интернета – ARPANET (англ. Advanced Research Projects Agency Network).

Появление первого микропроцессора в 1971 году произвело революцию в области цифровой электроники и вычислительных машин. Микропроцессор разрабатывался как элемент, способный заменить большое количество микросхем в составе процессорной платы ЭВМ. В результате в одном корпусе были скомпонованы счетчики, регистры, логические блоки и другие элементы [1].

Для получения готового компьютера к микропроцессору следовало добавить устройства ввода/вывода, память и некоторые другие элементы. Через некоторое время такой элемент был создан и получил название «микроконтроллер» (рис. 1.2).

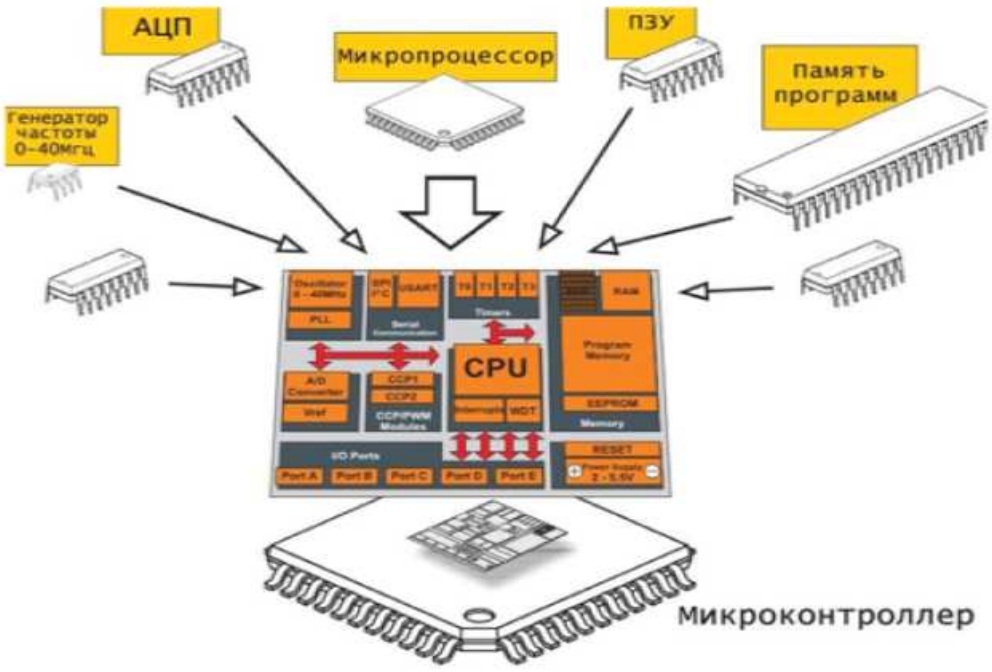

Рис. 1.2. Внутреннее устройство микроконтроллера

Микроконтроллер – микропроцессорная система, содержащая на одном кристалле процессорное ядро, долговременную и оперативную память, устройства ввода/вывода и предназначенная для построения систем управления, датчиков и других компактных устройств. В отличие от микропроцессора микроконтроллер обладает меньшей универсальностью, так как к нему нельзя подключать произвольные элементы [2, 3].

Разработчику доступно только то, что расположено внутри корпуса. Тем не менее эти микропроцессорные устройства получили огромное распространение. Количество производимых микроконтроллеров составляет 90 % от всех выпускаемых микропроцессоров [4, 37].

Сегодня микроконтроллеры используются во всей бытовой технике, промышленных установках и станках, датчиках, регуляторах и иных приложениях, где не требуется большая вычислительная мощность (рис. 1.3, 1.4). Например, в каждом современном автомобиле имеется несколько микроконтроллеров, управляющих двигателем, подвеской, кондиционером, акустической системой, приборной панелью.

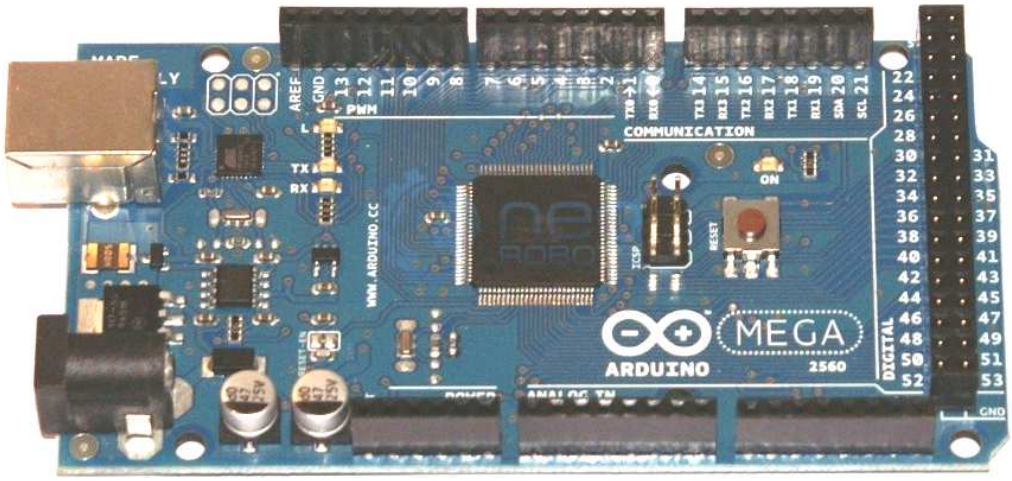

Рис. 1.3. Внешний вид Arduino

Первый патент на однокристальную микроЭВМ был выдан в 1971 году инженерам М. Кочрену и Г. Буну, сотрудникам американской Texas Instruments. Именно они предложили на одном кристалле разместить не только процессор, но и память с устройствами ввода/вывода.

Рис. 1.4. Микроконтроллер на плате бытовой техники

Фирма Intel внесла огромный вклад в развитие микроконтроллеров. Ее основатели – Роберт Нойс, Гордон Мур и Эндрю Гроув. Texas Instruments была основана в 1968 году [1].

До определенных пор фирма производила полупроводниковые запоминающие устройства. Первым была МС «3101» 64-го разряда (рис. 1.5).

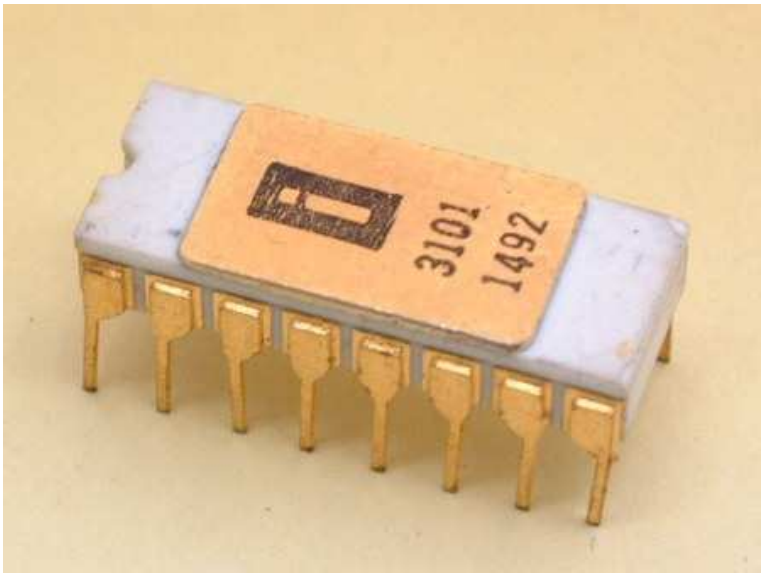

Рис. 1.5. Микропроцессор МС «3101»

Следующим было изобретение МС «4004» – микропроцессора с 2300 полупроводниковыми транзисторами в своем составе. По производительности он не хуже, чем ENIAC, а размером – меньше ладони, т. е. размер 4004-го микропроцессора был на много порядков меньше [1 – 5].

В 1976 году американская фирма Intel выпускает микроконтроллер i8048, в 1978 году фирма Motorola изготовила свой первый микроконтроллер MC6801, совместимый по системе команд с выпущенным ранее микропроцессором MC6800 (рис. 1.6, 1.7).

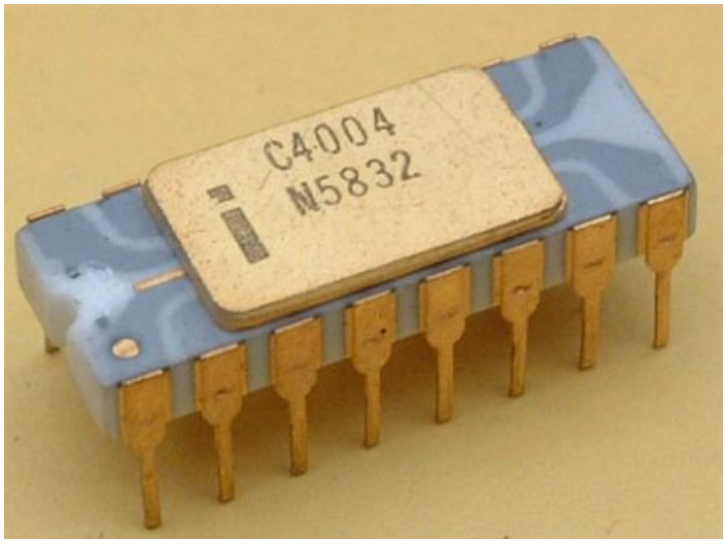

Рис. 1.6. Микропроцессор MC «4004»

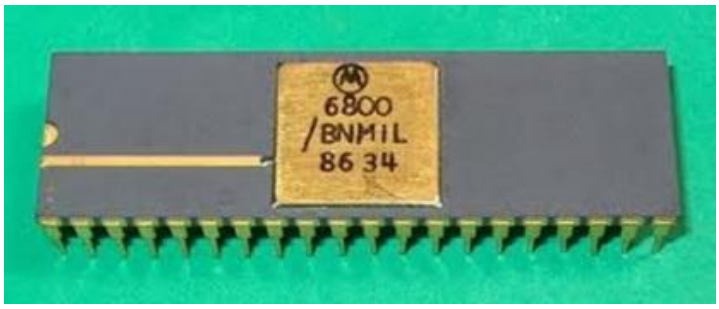

Рис. 1.7. Микропроцессор MC6800

Через четыре года, в 1980 году, Intel выпускает следующий микроконтроллер i8051 (рис. 1.8).

Рис. 1.8. Микроконтроллер i8051

Удачный набор периферийных устройств, возможность гибкого выбора внешней или внутренней программной памяти и приемлемая цена обеспечили этому микроконтроллеру успех на рынке. С точки зрения технологии микроконтроллер i8051 являлся для своего времени очень сложным изделием – в кристалле было использовано 128 тыс. транзисторов, что в четыре раза превышало количество транзисторов в 16-разрядном микропроцессоре i8086.

В СССР велись разработки оригинальных микроконтроллеров, также осваивался выпуск клонов наиболее удачных зарубежных образцов. В 1979 году в СССР НИИ ТТ разработали однокристальную 16-разрядную ЭВМ К1801ВЕ1, микроархитектура которой получила название «Электроника НЦ» [2, 3, 5, 6] (рис. 1.9).

Характеристика	К1801ВЕ1 (НЦ-80Т)	К586ВЕ1 (С5-31)
Разрядность данных, бит	1, 8, 16, 32	1, 8, 16
Разрядность АЛУ, бит	16	16
Число команд	404	132
ОЗУ, бит	128×16	128×18
ПЗУ, бит	1К×16	1К×16
Время сложения, мкс	3,1	2,0
Уровней прерываний	5	3
Ввод-вывод	16 бит магистраль, 32 бит посл. прогр. канал вв/вывода	16 бит магистраль, 6 каналов вв/вывода 8 бит посл. канал

Рис. 1.9. МикроЭВМ «К1801ВЕ1» и ее характеристики

Intel 4004 – 4-битный микропроцессор, разработанный корпорацией Intel и выпущенный 15 октября 1971 года. Эта микросхема считается первым в мире коммерчески доступным однокристальным микропроцессором (рис. 1.10).

Intel 8080 – 8-битный микропроцессор, выпущенный в 1974 году. Обеспечивал десятикратный прирост вычислительной производительности в сравнении с предыдущим процессором (рис. 1.11).

Благодаря этому устройству инженерное сообщество восприняло идею микропроцессоров и чип спровоцировал бум персональных компьютеров.

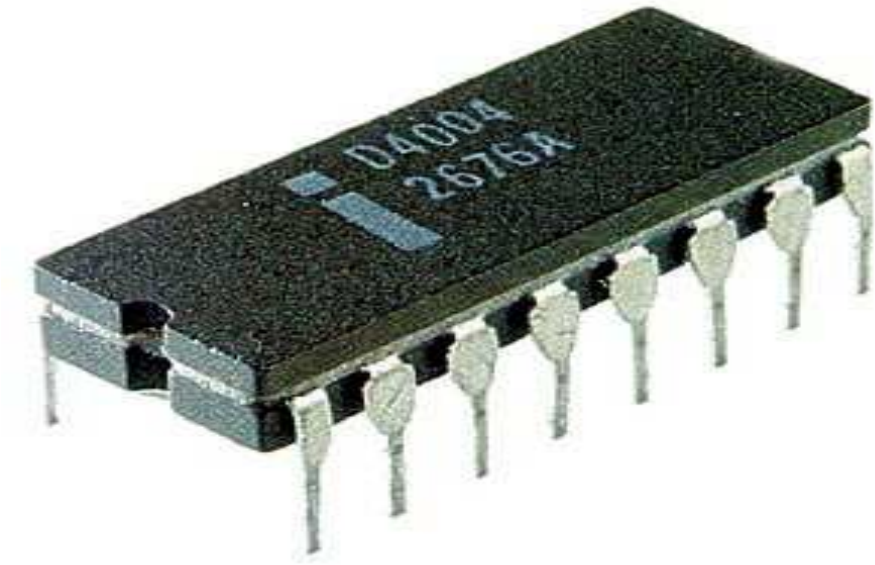

Рис. 1.10. 4-битный микропроцессор Intel 4004

Рис. 1.11. 8-битный микропроцессор Intel 8080

Intel 8048 – первый в мире микроконтроллер был выпущен в конце 70-х годов XX в.; получил широкое распространение благодаря использованию его в клавиатурах персональных компьютеров и в игровых приставках (рис. 1.12).

Рис. 1.12. Микроконтроллер Intel 8048

Intel 8051 – микроконтроллер второго поколения был выпущен в 1980 году (рис. 1.13). Благодаря удачной архитектуре и системе команд стал фактически промышленным стандартом. Выпускается до сих пор известными корпорациями Америки, Кореи и Японии.

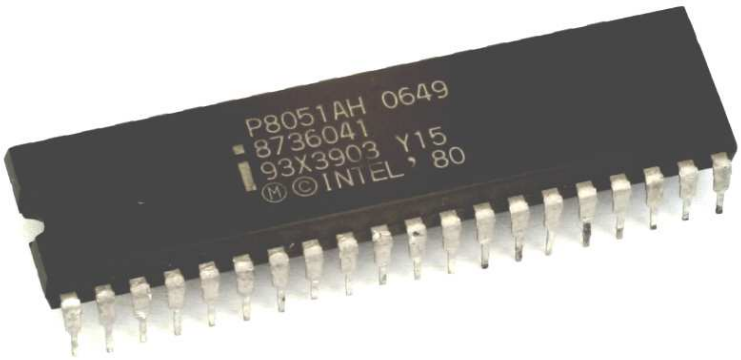

Рис. 1.13. Микроконтроллер второго поколения Intel 8051

При проектировании микроконтроллеров приходится соблюдать компромисс между размерами и стоимостью, с одной стороны, и гибкостью и производительностью – с другой. Для разных приложений оптимальное соотношение этих и других параметров может различаться очень сильно. Поэтому существует огромное количество типов микроконтроллеров, отличающихся архитектурой процессорного модуля, размером и типом встроенной памяти, набором периферийных устройств, типом корпуса и т. д. [1 – 6].

В отличие от обычных компьютерных микропроцессоров в микроконтроллерах часто используется гарвардская архитектура памяти, т. е. раздельное хранение данных и команд в ОЗУ и ПЗУ соответственно.

Кроме ОЗУ микроконтроллер может иметь встроенную энергонезависимую память для хранения программы и данных. Многие модели контроллеров вообще не имеют шин для подключения внешней памяти.

Наиболее дешевые типы памяти допускают лишь однократную запись либо хранимая программа записывается в кристалл на этапе изготовления (конфигурацией набора технологических масок). Такие устройства подходят для массового производства в тех случаях, когда программа контроллера не будет обновляться. Другие модификации контроллеров обладают возможностью многократной перезаписи программы в энергонезависимой памяти.

Неполный список периферийных устройств [3], которые могут использоваться в микроконтроллерах, включает в себя:

1) универсальные цифровые порты, которые можно настраивать как на ввод, так и на вывод;
2) различные интерфейсы ввода/вывода;
3) аналого-цифровые и цифроаналоговые преобразователи;
4) компараторы;
5) широтно-импульсные модуляторы (ШИМ-контроллер);
6) таймеры;
7) контроллеры бесколлекторных двигателей, в том числе шаговых;
8) контроллеры дисплеев и клавиатур;
9) радиочастотные приемники и передатчики;
10) массивы встроенной флеш-памяти;
11) встроенные тактовый генератор и сторожевой таймер.

Появление и развитие микроконтроллеров полностью изменило возможности современной техники. Она стала более функциональной, умной и при этом дешевой. Пример восьмибитного микроконтроллера ATmega приведен на рис. 1.4. Создавать многие устройства стало доступно не выходя из дома. В будущем применение микроконтроллеров будет только увеличиваться и вполне возможно, что

в скором времени они составят реальную конкуренцию на рынке персональных компьютеров.

Рис. 1.14. Восьмибитный микроконтроллер ATmega

1.2. Архитектура микроконтроллеров

Разработчиком архитектуры первого микропроцессора стал Тед Хофф, системы команд – Стен Мейзор, а Федерико Феджин спроектировал кристалл. Но изначально компания Intel не владела всеми правами на этот чип и, заплатив 60 000 дол. компании Busicom, которая вскоре обанкротилась, получила полные права.

Для популяризации и внедрения новых технологий Intel вела как рекламную, так и образовательную кампанию. Впоследствии и другие производители электроники объявляли о создании подобных устройств.

В последние десятилетия наиболее распространены следующие виды микроконтроллеров [4]:

1) 8-битные микроконтроллеры PIC фирмы Microchip Technology и AVR фирмы Atmel;

2) 16-битные MSP430 фирмы TI;

3) 32-битные микроконтроллеры архитектуры ARM, на базе которой выпускается масса различных продуктов, она продается разработчиками различным фирмам.

Различия микроконтроллеров

Микроконтроллеры можно разделить по таким критериям:

1) разрядность;
2) система команд;
3) архитектура памяти.

Разрядность – это длина одного слова, обрабатываемого контроллером или процессором. Чем она больше, тем быстрее микроконтроллер может обработать большие массивы данных, но такой подход не всегда справедлив. Для каждой задачи выдвигаются индивидуальные требования как по скорости, так и способу обработки, например, применение 32-разрядного микроконтроллера ARM может быть необосновано в случае работы с 8-битными данными. Трудности могут возникнуть как по удобству написания программы и обработки информации, так и по себестоимости микроконтроллера.

По разрядности современные микроконтроллеры можно разделить:

1) на 8-битные;
2) 16-битные;
3) 32-битные;
4) 64-битные.

Однако по статистике на 2017 год стоимость 32-битных контроллеров активно снижается, и если так будет продолжаться и далее, то они будут дешевле простейших PIC-контроллеров при наличии гораздо большего набора функций [4, 5, 37].

Приведем классификацию микроконтроллеров по типу системы команд [4, 6].

1. **RISC-архитектура**, или сокращенная система команд. Ориентирована на быстрое выполнение базовых команд за один, реже – два машинных цикла, а также имеет большое количество универсальных регистров и более длинный способ доступа к постоянной памяти. Архитектура характерна для систем под управление UNIX.

2. **CISC-архитектура**, или полная система команд. Характерна прямая работа с памятью, имеет большее число команд, малое количество регистров (ориентирована на работу с памятью), длительность команд – от одного до четырех машинных циклов. Пример – процессоры Intel. Отличие архитектур системы команд приведено в табл. 1.1.

Таблица 1.1

Отличие архитектур системы команд

CISC-архитектура	RISC-архитектура
1. Многобайтовые команды	1. Однобайтовые команды
2. Малое количество регистров	2. Большое количество регистров
3. Сложные команды	3. Простые команды
4. Одна команда за один цикл	4. Несколько команд за один цикл
5. Одно исполнительное устройство	5. Несколько исполнительных устройств

Классификация микроконтроллеров по типу памяти [4 – 6]:

1. **Архитектура фон Неймана** – основная особенность в том, что присутствует общая область памяти для команд и данных. При работе с такой архитектурой в результате ошибки программиста данные могут записаться в область памяти программ и дальнейшее выполнение программы станет невозможным. Одновременная пересылка данных и выборка команды не может осуществляться одновременно по тем же причинам. Подобная архитектура впервые была разработана в 1945 году (рис. 1.15).

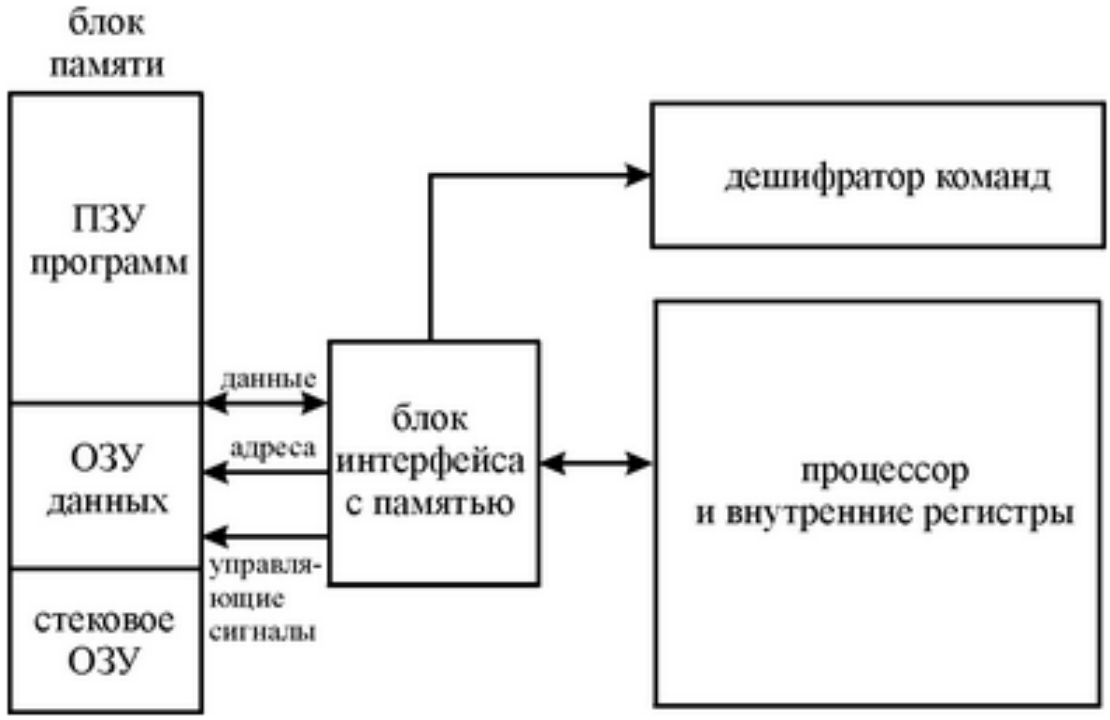

Рис. 1.15. Структура компьютера с Принстонской архитектурой

2. **Гарвардская архитектура** – основное отличие от архитектуры фон Неймана заключается в раздельной памяти данных и программ. Впервые использовалась на компьютерах семейства Mark. Разработана в 1944 году [4] (рис. 1.16).

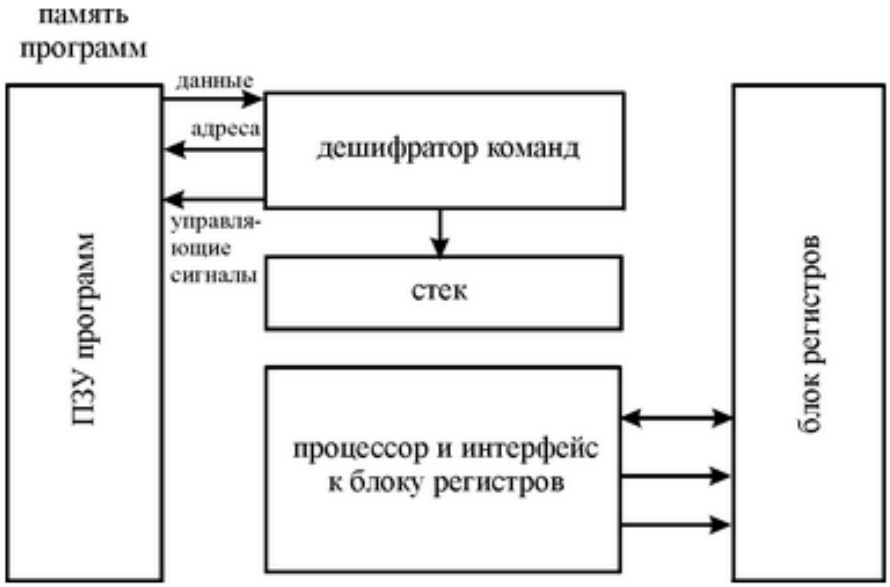

Рис. 1.16. Структура компьютера с Гарвардской архитектурой

В результате внедрения микропроцессорных систем размеры устройств уменьшились, а функционал увеличился. Выбор архитектуры, разрядности, системы команд, структуры памяти влияет на конечную стоимость устройства, поскольку при единичном производстве разница в цене может быть незначительна, но при тиражировании – более чем ощутимой.

1.3. Цифроаналоговые и аналого-цифровые преобразователи

Одним из бурно развивающихся направлений в электронной промышленности является создание полностью интегрированных систем управления, включающих блоки для аналоговой и цифровой обработки сигнала. Большой спрос на сложные системы управления с высокоскоростными и точными аналоговыми и цифровыми преобразованиями определяет их развитие. Необходимым компонентом таких систем служат блоки для оцифровки аналоговых сигналов с целью их дальнейшей обработки [7].

В электронике под аналого-цифровыми преобразователями (АЦП) (в переводе с англ. от ADC – analog-to-digital converter) понимают устройство, которое конвертирует аналоговый сигнал (например, ток или напряжение) в цифровой код (двоичную форму). В реальном мире большинство сигналов являются аналоговыми, но все микроконтроллеры и микропроцессоры способны понимать только двоичные (бинарные) сигналы – 0 или 1, т. е., чтобы заставить микроконтроллер понимать аналоговые сигналы, необходимо конвертировать их в цифровую форму – это и делает АЦП.

Существуют различные типы АЦП, каждый тип удобен для конкретных приложений. Наиболее популярные АЦП используют такие типы аппроксимаций, как приближенная, последовательная и дельта-аппроксимация, но вместе с тем следует помнить о том, что существуют и другие типы микроконтроллеров, у которых нет собственных АЦП, – в этом случае необходимо использовать внешний АЦП [7]. Как правило, внешние АЦП сейчас выпускаются в виде одной микросхемы.

1. Самые дешевые АЦП – с последовательной аппроксимацией. В данном случае для каждого фиксированного аналогового уровня последовательно формируется серия соответствующих им цифровых кодов. Внутренний счетчик используется для их сравнения с аналоговым сигналом после конверсии. Генерация цифровых кодов останавливается, когда соответствующий им аналоговый уровень становится чуть больше, чем аналоговый сигнал на входе АЦП. Этот цифровой код и будет представлять собой конвертированное значение аналогового сигнала [8].

2. Сигма-дельта модулятор (рис. 1.17) можно условно рассматривать как синхронный преобразователь напряжение-частота, и теоретически возможно подсчитать удельное количество единиц в этом цифровом потоке, что и будет цифровым кодом простейшего сигма-дельта АЦП. Однако на практике такой метод не применяется из-за большого количества требуемых выборок. На практике используется цифровая фильтрация шума квантования, который благодаря структуре сигма-дельта модулятора имеет спад в области низких частот, причем спад имеет большую крутизну в модуляторах более высокого порядка. Таким образом, соотношение сигнал/шум растет не только за счет избыточной дискретизации, но и благодаря ограничению шума (англ. noise shaping) в области частот, содержащей полезный сигнал [8].

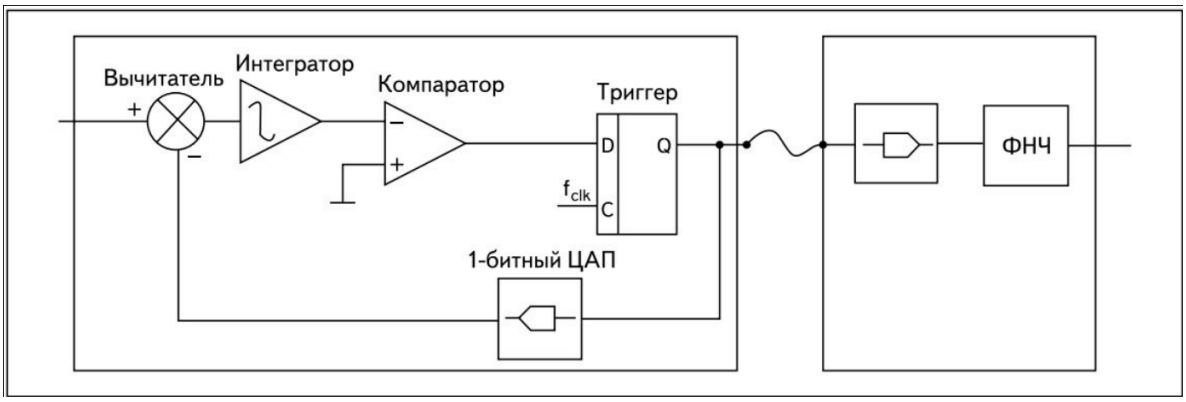

Рис. 1.17. Сигма-дельта модулятор

Аналого-цифровое преобразование (АЦП, ADC) содержит три фазы: дискретизацию по времени, квантование по уровню, цифровое, двоичное кодирование.

Микроконтроллеры с интегрированными АЦП и цифро-аналоговыми преобразователями (ЦАП) являются удобным решением для систем сбора и обработки данных от многих датчиков. Уступая чисто аналоговой обработке только в быстродействии, такой микроконтроллер отличается большой функциональной гибкостью и точностью благодаря наличию обширной аналоговой и цифровой периферии, что значительно упрощает разработку конечного устройства.

В настоящее время все чаще применяют «интеллектуальные датчики». Такой датчик имеет встроенный микропроцессор, выполняющий некоторую обработку сигнала, и поэтому может давать более точные показания благодаря применению числовых вычислений для компенсации нелинейностей чувствительного элемента или температурной зависимости. В круг возможностей некоторых приборов входят измерение нескольких параметров и пересчет их в одно измерение (например, объемный расход, температуру и давление – в массовый расход, так называемые многопараметрические датчики), функции встроенной диагностики, автоматическая калибровка [8].

Некоторые интеллектуальные приборы (например, семейство приборов Rosemount SMART FAMILY) позволяют посылать в канал передачи аналоговый и цифровой сигналы. В случае одновременной трансляции обоих видов сигналов аналоговый используется для трансляции значения измеренного параметра, а цифровой – для функций настройки, калибровки, а также позволяет считывать измеряемый параметр. Эти устройства обеспечивают преимущества цифровой связи и в то же время сохраняют совместимость и надежность аналоговых средств, которые требуются для существующих систем [7, 8].

1.4. Интерфейсы микроконтроллеров

К основным типам интерфейсов, встречающихся в микроконтроллерах, относятся UART/USART, SPI, JTAG. Именно с помощью этих интерфейсов осуществляются связь микроконтроллера с персональным компьютером, обмен данными с другими цифровыми устройствами и программирование самого микроконтроллера.

Последовательный интерфейс UART/USART

Универсальный асинхронный или универсальный синхронно/асинхронный приемопередатчик (Universal Synchronous/ Asynchronous Receiver and Transmitter – UART или USART) – удобный и простой последовательный интерфейс для организации информационного канала обмена микроконтроллера с внешним миром. Способен работать в дуплексном режиме (одновременная передача и прием данных); поддерживает протокол стандарта RS-232, что обеспечивает возможность организации связи с персональным компьютером [3].

Изначально использовался в компьютерах для большинства периферийных устройств, таких как плоттер, удаленный принтер, мышь, внешний модем и т. д. До настоящего времени для последовательной связи IBM PC-совместимых компьютеров используются адаптеры с интерфейсом RS-232C (новое название EIA-232D). В современном IBM PC-совместимом компьютере может использоваться до четырех последовательных портов, имеющих логические имена соответственно COM1, COM2, COM3 и COM4. Основой последовательного адаптера является микросхема UART (Universal Asynchronous Receiver/Transmitter) – универсальный асинхронный приемопередатчик. Обычно применяется микросхема UART 16550A, имеющая 16-символьный буфер на прием и передачу и, кроме того, использующая несколько каналов прямого доступа в память DMA.

При передаче микросхема UART преобразует параллельный код в последовательный и передает его побитно в линию, обрамляя исходную последовательность битами старта, останова и контроля. При приеме данных UART преобразует последовательный код в параллельный (разумеется, опуская служебные символы). Непременным условием правильной передачи (приема) является одинаковая ско-

рость работы приемного и передающего UART, что обеспечивается стабильной частотой кварцевого резонатора. Основным преимуществом последовательной передачи следует считать возможность пересылки данных на большие расстояния, как правило, не менее 30 м. В IBM PC-совместимых персональных компьютерах из 25 сигналов, предусмотренных стандартом RS-232, используются в соответствии с EIA только 9; таким образом, в данном интерфейсе, как правило, применяются 9-контактные разъемы типа DB-Shell. В современных компьютерах UART и COM порт уже не применяются напрямую, но они получили вторую жизнь для связи с различными нестандартными внешними устройствами, в число которых вошли и устройства на микроконтроллерах. Аппаратная часть при этом стала значительно проще для связи микроконтроллеров друг с другом; подключение UART осуществляется по трем линиям: RXD – прием, TXD – передача и GND – общий (минус) [3, 6] (рис. 1.18).

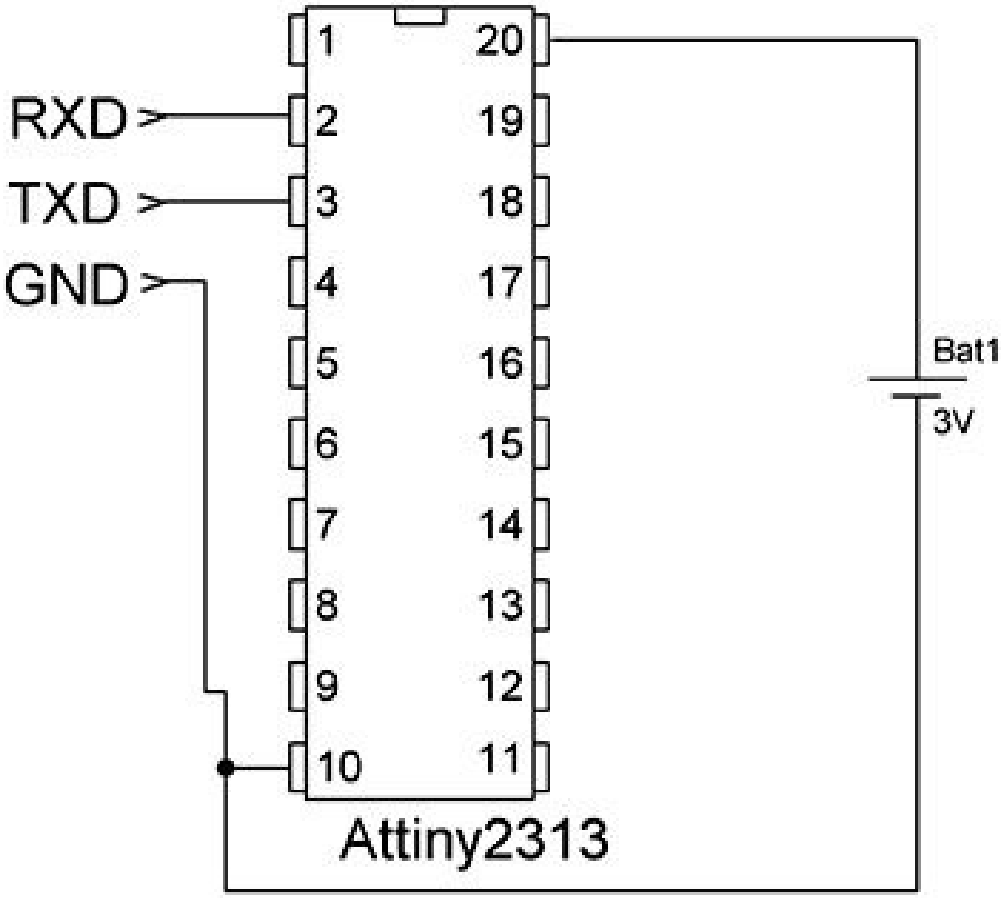

Рис. 1.18. Схема подключения микроконтроллера Attiny2313

Подключать UART надо, так сказать, «наоборот»: RXD к TXD, а TXD к RXD (как на рис. 1.19).

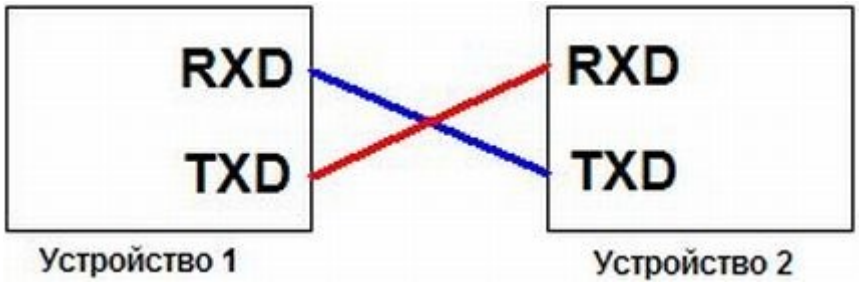

Рис. 1.19. Схема соединения двух устройств (общий минус не показан)

С помощью UART также можно связать микроконтроллер и компьютер, но есть одна проблема: у UART интерфейса логические уровни 0 и +5 В, а в компьютере логические уровни в интерфейсе RS-232 могут быть от –25 до –3 В и от +3 до +25 В. Для этого применяют специальный преобразователь уровней на микросхеме MAX232 (рис. 1.20).

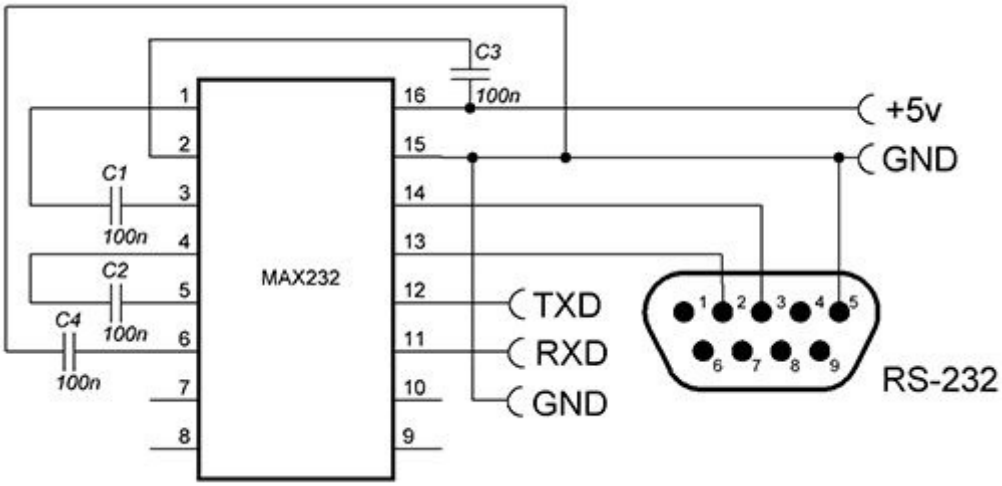

Рис. 1.20. Преобразователь уровней

Все сигналы UART передаются специально выбранными уровнями, обеспечивающими высокую помехоустойчивость связи. Отметим, что данные передаются в инверсном коде (логической единице соответствует низкий уровень, логическому нулю – высокий уровень). Собственно данные (5, 6, 7 или 8 бит) сопровождаются стартовым битом, битом четности и одним или двумя стоповыми битами (рис. 1.21). Получив стартовый бит, приемник выбирает из линии би-

ты данных через определенные интервалы времени. Очень важно, чтобы тактовые частоты приемника и передатчика были одинаковыми, допустимое расхождение – не более 10 %. Скорость передачи по RS-232C может выбираться из ряда: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 бит/с.

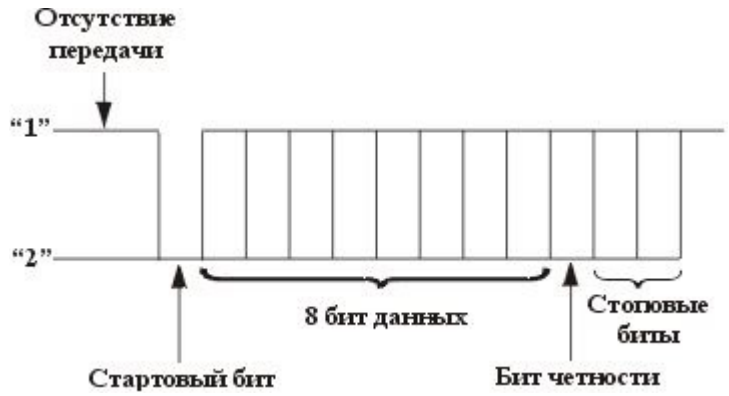

Рис. 1.21. Формат данных

Последовательный периферийный интерфейс SPI

Последовательный периферийный трехпроводный интерфейс SPI (Serial Peripheral Interface) предназначен для организации обмена данными между двумя устройствами. С его помощью может осуществляться обмен данными между микроконтроллером и различными устройствами, такими как цифровые потенциометры, ЦАП/АЦП, FLASH-ПЗУ и др. С помощью этого интерфейса удобно производить обмен данными между несколькими микроконтроллерами AVR. Кроме того, через интерфейс SPI может осуществляться программирование микроконтроллера [8, 36].

Изначально он был придуман компанией Motorola, а в настоящее время используется в продукции многих производителей. Его наименование является аббревиатурой от Serial Peripheral Bus, что отражает его предназначение – шина для подключения внешних устройств. Шина SPI организована по принципу ведущий – подчиненный. В качестве ведущего шины обычно выступает микроконтроллер, но им также может быть программируемая логика, DSP-контроллер или специализированная интегральная схема (ИС). Подключенные к ведущему шины образуют подчиненные шины. В их роли выступают различного рода микросхемы, в том числе запоминающие устройства (EEPROM, Flash-память, SRAM), часы реального

времени (RTC), АЦП/ЦАП, цифровые потенциометры, специализированные контроллеры и др. [8].

Главным составным блоком интерфейса SPI (рис. 1.22) является обычный сдвиговый регистр, сигналы синхронизации и ввода/вывода битового потока которого и образуют интерфейсные сигналы.

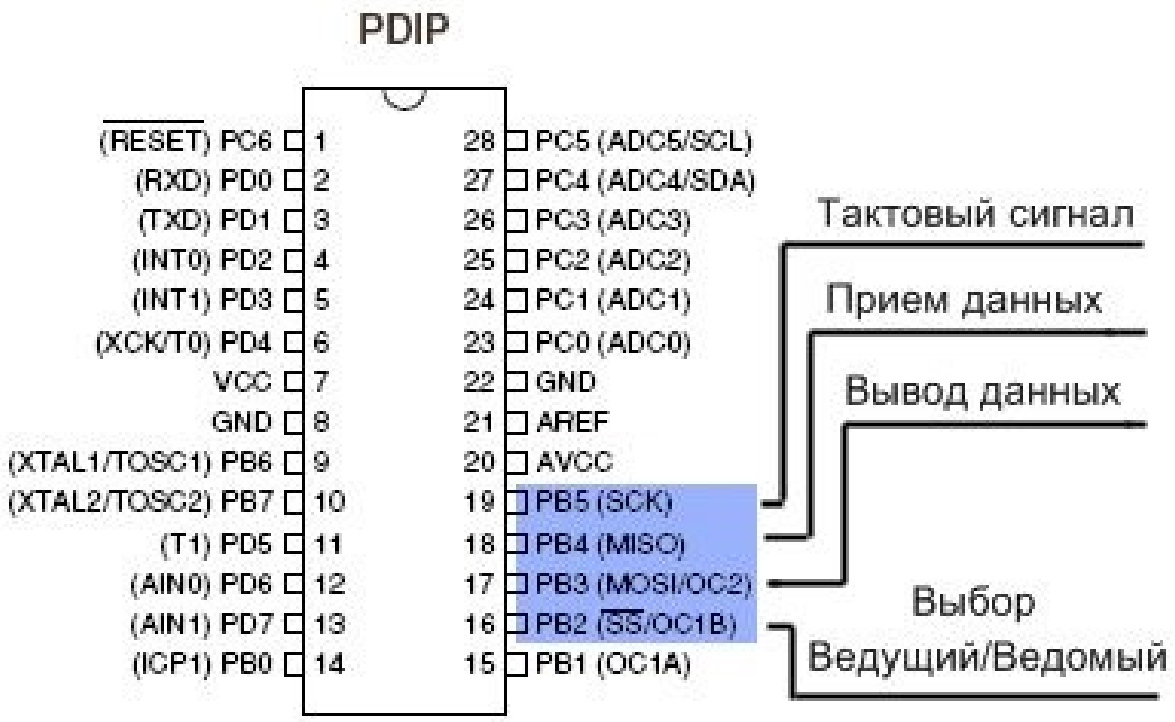

Рис. 1.22. Расположение выводов интерфейса SPI на микроконтроллере AtMega8

Таким образом, протокол SPI правильнее назвать не протоколом передачи данных, а протоколом обмена данными между двумя сдвиговыми регистрами, каждый из которых одновременно выполняет и функцию приемника, и функцию передатчика. Непременным условием передачи данных по шине SPI является генерация сигнала синхронизации шины. Этот сигнал имеет право генерировать только ведущий шины, и от этого сигнала полностью зависит работа подчиненного шины.

Электрическое подключение

Существует три типа подключения к шине SPI, в каждом из которых участвуют четыре сигнала (рис. 1.23). Ведущий шины передает данные по линии MOSI синхронно со сгенерированным им же сигна-

лом SCLK, а подчиненный захватывает переданные биты данных по определенным фронтам принятого сигнала синхронизации. Одновременно с этим подчиненный отправляет свою посылку данных. Представленную схему можно упростить исключением линии MISO, если используемая подчиненная ИС не предусматривает ответную передачу данных или в ней нет потребности. Одностороннюю передачу данных можно встретить у таких микросхем, как ЦАП, цифровые потенциометры, программируемые усилители и драйверы.

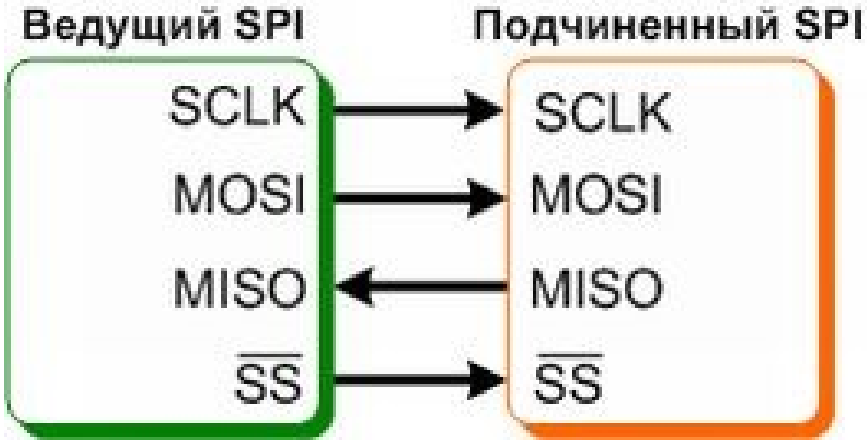

Рис. 1.23. Простейшее подключение к шине SPI

Таким образом, рассматриваемый вариант подключения подчиненной ИС требует 3 или 4 линии связи. Для того чтобы подчиненная ИС принимала и передавала данные, помимо наличия сигнала синхронизации необходимо также, чтобы линия SS была переведена в низкое состояние. В противном случае подчиненная интегральная схема будет неактивна. Когда используется только одна внешняя ИС, может возникнуть соблазн исключения и линии SS за счет жесткой установки низкого уровня на входе выбора подчиненной микросхемы. Такое решение крайне нежелательно и может привести к сбоям или невозможности передачи данных, так как вход выбора микросхемы служит для перевода ИС в ее исходное состояние и иногда инициирует вывод первого бита данных.

При необходимости подключения к шине SPI нескольких микросхем используется либо независимое (параллельное) подключение, либо каскадное (последовательное) (рис. 1.24, 1.25).

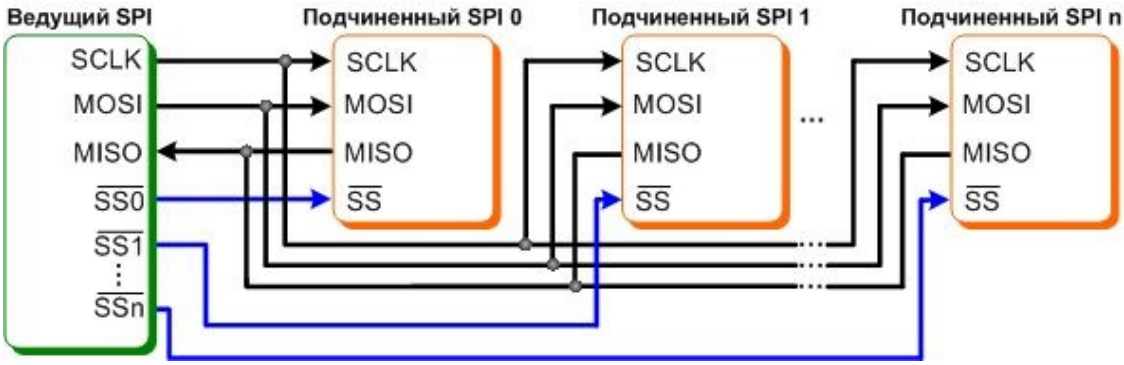

Рис. 1.24. Независимое подключение к шине SPI

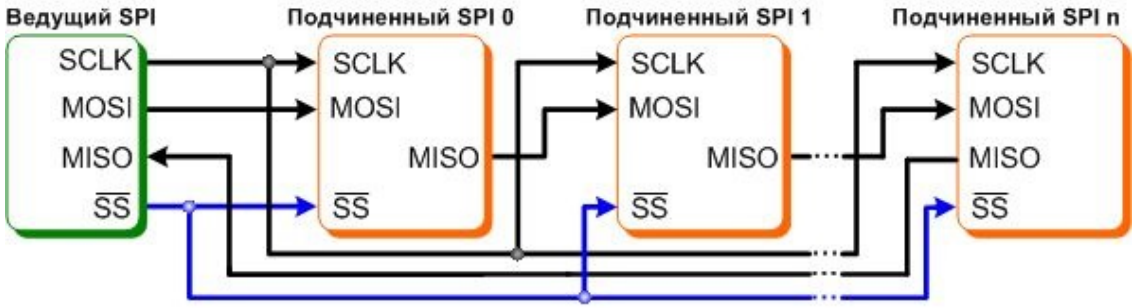

Рис. 1.25. Каскадное подключение к шине SPI

Независимое подключение более распространено, так как достигается при использовании любых SPI-совместимых микросхем. Здесь все сигналы, кроме выбора микросхем, соединены параллельно, а ведущий шины переводом того или иного сигнала SS в низкое состояние задает, с какой подчиненной ИС он будет обмениваться данными. Главным недостатком такого подключения является необходимость в дополнительных линиях для адресации подчиненных микросхем (общее число линий связи равно 3 + n, где n – количество подчиненных микросхем) [7, 8].

Каскадное включение избавлено от этого недостатка, так как здесь из нескольких микросхем образуется один большой сдвиговый регистр. Для этого выход передачи данных одной ИС соединяется со входом приема данных другой. Входы выбора микросхем соединены параллельно, и таким образом общее число линий связи сохранено равным четырем. Однако использование каскадного подключения возможно только в том случае, если его поддержка указана в документации на используемые микросхемы. Для того чтобы выяснить это, важно знать, что такое подключение называется по-английски daisy-chaining.

Протокол передачи

Протокол передачи по интерфейсу SPI предельно прост и по сути идентичен логике работы сдвигового регистра, которая заключается в выполнении операции сдвига и соответственно побитного ввода и вывода данных по определенным фронтам сигнала синхронизации. Установка данных при передаче и выборка при приеме всегда выполняются по противоположным фронтам синхронизации. Это необходимо для гарантирования выборки данных после надежного их установления. Если учесть, что в качестве первого фронта в цикле передачи может выступать нарастающий или падающий фронт, то возможны всего четыре варианта логики работы интерфейса SPI [8]. Эти варианты получили название режимов SPI и описываются двумя параметрами:

CPOL – исходный уровень сигнала синхронизации (если CPOL = 0, то линия синхронизации до начала цикла передачи и после его окончания имеет низкий уровень, т. е. первый фронт нарастающий, а последний – падающий; если CPOL = 1, – высокий, т. е. первый фронт падающий, а последний – нарастающий);

CPHA – фаза синхронизации; от этого параметра зависит, в какой последовательности выполняются установка и выборка данных (если CPHA = 0, то по переднему фронту в цикле синхронизации будет выполняться выборка данных, а затем, по заднему фронту – установка данных; если же CPHA = 1, то установка данных будет выполняться по переднему фронту в цикле синхронизации, а выборка – по заднему). Информация по режимам SPI обобщена на рис. 1.26.

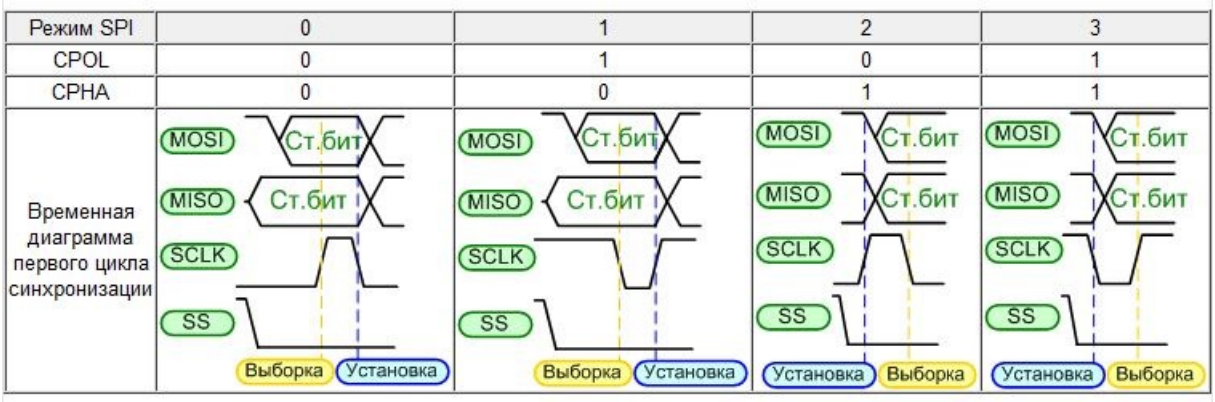

Рис. 1.26. Режимы SPI

Ведущая и подчиненная микросхемы, работающие в различных режимах SPI, являются несовместимыми, поэтому перед выбором подчиненных микросхем важно уточнить, какие режимы поддерживаются ведущим шины. Аппаратные модули SPI, интегрированные в микроконтроллеры, в большинстве случаев поддерживают возможность выбора любого режима SPI и поэтому к ним возможно подключение любых подчиненных SPI-микросхем (относится только к независимому варианту подключения). Кроме того, протокол SPI в любом из режимов легко реализуется программно.

Протокол отладки

JTAG (сокращение от англ. Joint Test Action Group) – название рабочей группы по разработке стандарта IEEE 1149. Позднее это сокращение стало прочно ассоциироваться с разработанным этой группой специализированным аппаратным интерфейсом на базе стандарта IEEE 1149.1. Официальное название стандарта **Standard Test Access Port and Boundary-Scan Architecture**. Интерфейс предназначен для подключения сложных цифровых микросхем или устройств уровня печатной платы к стандартной аппаратуре тестирования и отладки [9].

На текущий момент интерфейс стал промышленным стандартом. Практически все сколько-нибудь сложные цифровые микросхемы оснащаются этим интерфейсом:

- для выходного контроля микросхем при производстве;
- тестирования собранных печатных плат;
- прошивки микросхем с памятью;
- отладочных работ при проектировании аппаратуры и программного обеспечения.

Метод тестирования, реализованный в стандарте, получил название Boundary Scan (граничное сканирование). Название отражает первоначальную идею процесса: в микросхеме выделяются функциональные блоки, входы которых можно отсоединить от остальной схемы, подать заданные комбинации сигналов и оценить состояние выходов каждого блока. Весь процесс производится специальными командами по интерфейсу JTAG (рис. 1.27), при этом никакого физического вмешательства не требуется. Разработан стандартный язык управления данным процессом – Boundary Scan Description Language (BSDL).

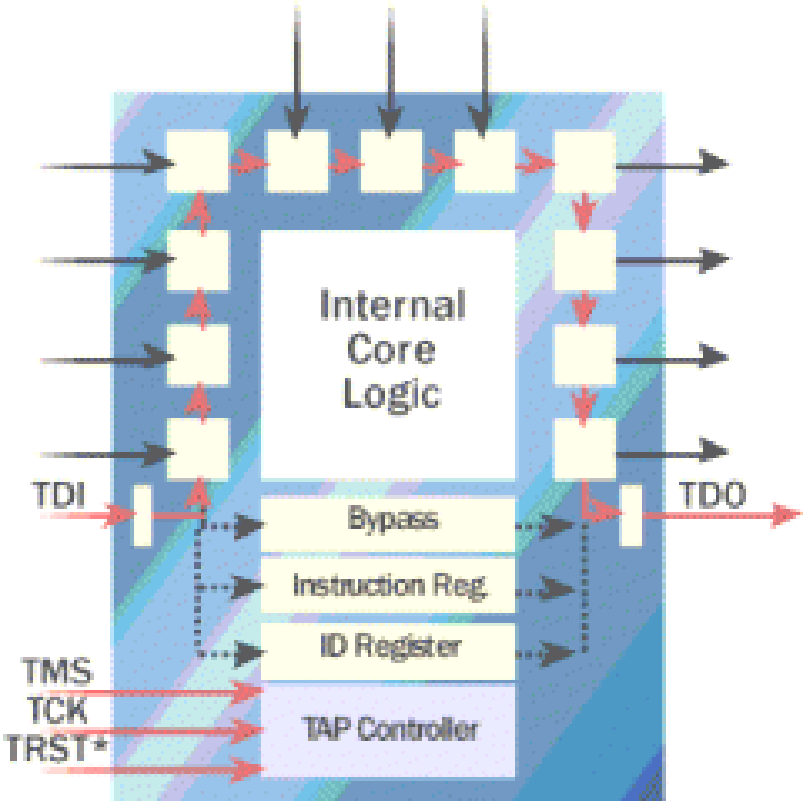

Рис. 1.27. JTAG-порт микросхемы
и ячейки периферийного сканирования

Стандарт предусматривает возможность подключения большого количества устройств (микросхем) через один физический порт (разъем) (рис. 1.28).

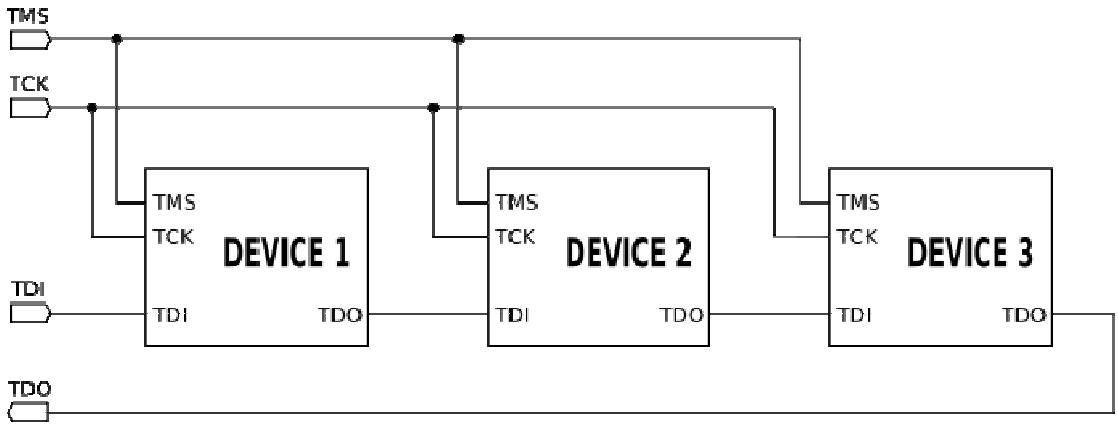

Рис. 1.28. Программирование нескольких устройств

Порт тестирования (TAP – Test Access Port) представляет собой четыре или пять выделенных выводов микросхемы: TCK, TMS, TDI, TDO и (опционально) TRST.

Функциональное назначение этих линий [9]:

1. TDI (test data input – «вход тестовых данных») – вход последовательных данных периферийного сканирования. Команды и данные вводятся в микросхему с этого вывода по переднему фронту сигнала TCK.

2. TDO (test data output – «выход тестовых данных») – выход последовательных данных. Команды и данные выводятся из микросхемы с этого вывода по заднему фронту сигнала TCK.

3. TCK (test clock – «тестовое тактирование») – тактирует работу встроенного автомата управления периферийным сканированием. Максимальная частота сканирования периферийных ячеек зависит от используемой аппаратной части и на данный момент ограничена 25 – 40 МГц.

4. TMS (test mode select – «выбор режима тестирования») – обеспечивает переход схемы в/из режима тестирования и переключение между разными режимами тестирования.

5. В некоторых случаях к перечисленным сигналам добавляется сигнал TRST для инициализации порта тестирования, что необязательно, так как инициализация возможна путем подачи определенной последовательности сигналов на вход TMS.

Работа средств обеспечения интерфейса JTAG подчиняется сигналам автомата управления, встроенного в микросхему. Состояния автомата определяются сигналами TDI и TMS порта тестирования. Определенное сочетание сигналов TMS и TCK обеспечивает ввод команды для автомата и ее исполнение.

Если на плате установлено несколько устройств, поддерживающих JTAG, они могут быть объединены в общую цепочку. Уникальной особенностью JTAG следует назвать возможность программирования не только самого микроконтроллера (или ПЛИС), но и подключенной к его выводам микросхемы флеш-памяти. Причем существует два способа программирования флеш-памяти с использованием JTAG: через загрузчик с последующим обменом данными, через память процессора либо через прямое управление выводами микросхемы [Там же].

1.5. Современные микроконтроллеры

1. Корпорация Nuvoton Technology (NTC) [4, 37] основана и зарегистрирована в Тайване. Она создавалась для привнесения инновационных решений в сферу аналоговых/смешанных сигналов на рынке полупроводников. Начав работу в качестве дочерней компании Winbond Electronics, Nuvoton Technology в 2008 году открыла свой офис и с этого момента присутствует на рынке микроэлектроники с собственным брендом. Между тем Nuvoton сохранила все связи с Winbond и унаследовала главное конкурентное преимущество – низкую стоимость памяти. На фондовой бирже Тайваня (TSE) компания присутствует с осени 2010 года.

Продукция Nuvoton выпускается на фабриках Winbond и TSMC, что гарантирует определенную независимость от внешних факторов на рынке. Компании не коснулись санкционные ограничения США и Европы на поставки своей продукции в Россию. Штаб-квартира, дизайн-центр и обе фабрики Nuvoton расположены в непосредственной близости друг от друга в научно-промышленном кластере г. Синьчу (Тайвань). Продукция Nuvoton удовлетворяет требованиям Европейской директивы об ограничении опасных веществ (RoHS) [4].

В составе Nuvoton три подразделения: микроконтроллеры и микросхемы для аудиорешения для облачных и вычислительных систем (включая персональные компьютеры), а также собственное производство пластин. В 2020 году Nuvoton приобрела полупроводниковый бизнес Panasonic (контроллеры питания и датчики для автоэлектроники, смартфонов и систем безопасности).

Для своих клиентов и партнеров Nuvoton предлагает современную продукцию с отличным соотношением цены и качества. Это достигается благодаря гибкому использованию технологий, расширенным возможностям дизайна и интеграции аналоговых и цифровых решений. Компания ценит долговременные отношения с партнерами и потребителями. Для улучшения региональной поддержки клиентов и повышения уровня сервиса Nuvoton основала дочерние компании в США, Китае, Израиле и Индии.

2. Компания WIZnet (Южная Корея) [Там же] занимается производством и поставкой уникальных аппаратных решений классов Embedded Internet и Embedded Ethernet. Микросхемы WIZnet приме-

няются для организации удаленного доступа, в системах IP-телефонии и IP-телевидения, в системах охраны, безопасности и контроля доступа, а также других приложений, использующих для коммуникации проводные каналы связи и сети Ethernet с развитым стеком протоколов TCP/IP.

Основной продукцией компании Wiznet являются интегральные микросхемы сетевых контроллеров: аппаратные Ethernet-мосты и специализированные микроконтроллеры. В последних на одном кристалле объединяются аппаратный блок Ethernet-моста со встроенными уровнями MAC и PHY и процессорное ядро ARM или MSC-51 [4].

3. Компания Megawin Technology Co., Ltd была основана в Тайване группой специалистов в 1999 году. Среди продукции Megawin можно выделить 32-битные микроконтроллеры (MCU) на базе Cortex-M0 серий MG32F02A и MG32F02U с АЦП и высокоточным внутренним генератором, 8-битные контроллеры на основе ядра MCS-51 (8051), а также USB-устройства с отличным соотношением цены и качества [2, 4].

Продукция Megawin широко применяется в автомобильной электронике (VAITRIX), системах управления питанием (Orion BMS), вендинговых аппаратах (ADONGING) и во многих других областях. Устройства Megawin могут использоваться практически во всех типах электронного оборудования: системах освещения, системах охраны и безопасности, промышленных приборах, вычислительной и медицинской технике.

В настоящее время Megawin экспортирует более 5 млн единиц MCU каждый месяц на рынок Кореи, Юго-Восточной Азии, Индии, России и Китая. Компания Megawin может осуществлять поставки микроконтроллеров, запрограммированных на заводе.

Megawin предоставляет комплексные инструменты для разработки, такие как ICE (внутрисхемный эмулятор), Writer и ISP (In-System Program), Programmer. Компания Megawin активно инвестирует в исследования и разработку, владеет 77 патентами, соответствует требованиям ISO 9001.

4. Американская фирма Silicon Labs (SiLabs), головной офис которой находится в Остине, штат Техас, была основана в 1998 году и в настоящее время является одним из мировых лидеров в разработке технологических и архитектурных решений микроэлектроники для

совместной обработки аналоговых и цифровых сигналов. Дополняя свою полупроводниковую продукцию профессиональным программным обеспечением и отладочными средствами, компания Silicon Labs предлагает законченные решения для реализации самых смелых инженерных идей для будущего информационно связанного мира [4].

Ведущая политика фирмы – стремление предоставить своим клиентам максимальные функциональные возможности в каждой микросхеме за счет высокой степени интеграции узлов и элементов на кристалле при одновременной минимизации размеров микросхемы. Silicon Labs выпускает широкий спектр продукции специального и общего назначения, которая используется в бытовой технике, индустриальной и автомобильной промышленности, в телекоммуникационных системах и офисном оборудовании. Помимо 8- и 32-разрядных микроконтроллеров Silicon Labs выпускает решения для управления питания и гальванической развязки, микросхемы для радиовещания и беспроводных телекоммуникаций, генераторы и генераторы управляемого напряжения, микросхемы для проводных коммуникаций и микросхемы встраиваемых проводных модемов ISOmodem.

5. Компания Microchip Technology была основана в 1991 году. Она предлагает широкий спектр цифровой и аналоговой продукции, позволяющий разработчикам найти все компоненты для своей задачи: микроконтроллеры и микропроцессоры, энергонезависимую память, микросхемы программируемой логики, компоненты для тактирования, интерфейсные решения, АЦП и ЦАП, датчики температуры и дыма, ультразвуковые решения, драйверы двигателей, высоковольтные интерфейсы и решения для управления питанием, расширители портов ввода/вывода, драйверы дисплеев и светодиодов, компоненты для беспроводной связи, микросхемы с блоками криптографии, сенсорные решения, а также операционные усилители и компараторы [2, 4].

Компания Microchip постоянно расширяет номенклатуру выпускаемой продукции, в том числе и за счет присоединения новых компаний. Приобретение Atmel в 2016 году позволило Microchip добавить в свою программу поставки микроконтроллеры AVR 8-бит, ARM 32-бит, микросхемы памяти, микросхемы с блоками криптографии, беспроводные компоненты, микросхемы для ответственных приложений (Aerospace).

К особенностям компании Microchip можно отнести то, что она не снимает с производства морально устаревшие компоненты. Благодаря этому любой из компонентов, заложенный заказчиком в какой-либо проект, будет доступен в течение всей жизни проекта. Эта политика будет распространена и на продукцию Atmel. Обозначения выпускавшихся ранее Atmel компонентов также будут сохранены без изменений, но теперь они будут поставляться на рынок под общим брендом Microchip.

Начиная с процессорного ядра микроконтроллеры различаются по следующим параметрам:

- тактовой частоте;
- корпусу;
- количеству линий ввода/вывода;
- напряжению питания;
- температурному диапазону;
- интерфейсу отладки;
- flash-памяти;
- ОЗУ;
- ШИМ;
- блокам АЦП.

Ядра микроконтроллеров выбираются в зависимости от задачи и целей, их различие обращается в плюсы или минусы. В табл. 1.2 приведены основные различия в параметрах процессного ядра.

Таблица 1.2

Основные различия в параметрах процессорного ядра

Процессорное ядро	Тактовая частота, МГц	Flash-память, Кб	ОЗУ, Кб	EEPROM*, байт
8051	11/24/36	8/16/32/64	1/4/64	0 – 255
ARM Cortex-M0 / M0+	24/36/42 – 48/50/72	4 – 256	2 – 96	
ARM Cortex-M23	48/64	16 – 128/512	8 – 16/96	
ARM Cortex-M4 / M4F	72/100/192	40 – 256/512 – 2048	16 – 32/128/160	
ARM926EJ-S	200 – 300	0	2048 – 131072	

* EEPROM (англ. Electrically Erasable Programmable Read-Only Memory) – электрически стираемое перепрограммируемое ПЗУ.

Основные различия в коммуникационных интерфейсах: разрядность АЦП, количество каналов преобразователя, разрядность обратного преобразователя и количество его каналов, присутствие датчика температуры, количество операционных усилителей и аналоговых компараторов, таймеры, наличие real time, количество каналов ШИМ, количество rc-генераторов, PLL и входов внешних резонаторов [2]. Характеристики современных микроконтроллеров приведены на рис. 1.29.

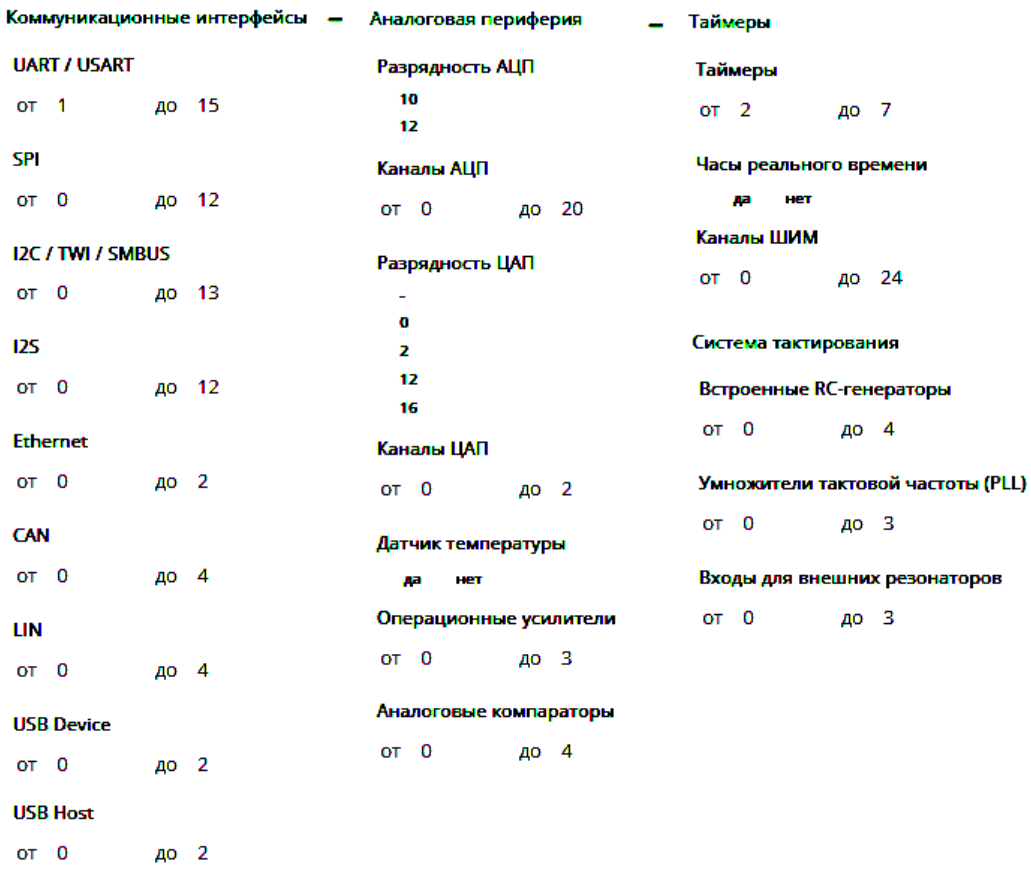

Рис. 1.29. Характеристики современных микроконтроллеров

1.6. Отладочные платы на базе микроконтроллеров

Сегодня существует множество уже готовых контроллерных плат с размещенными на них микроконтроллерами и средствами загрузки программ. Такие платы предназначены для оценки работы с микроконтроллером (называются оценочными платами) и могут работать в рамках законченного устройства.

Рассмотрим несколько популярных решений, базирующихся на микроконтроллерах известных производителей.

Плата, на которой установлен микроконтроллерный модуль ESP8266 с процессором Tensilica Xtensa L106, работающим на частоте 80 МГц (можно разогнать до 160 МГц), имеет 4 Мб флеш-памяти. Размер NodeMCU всего 6 × 3 см. NodeMCU умеет работать с локальной сетью или с Интернетом через Wi-Fi (рис. 1.30).

Рис. 1.30. NodeMCU (ESP8266)

Программы загружаются через разъем Micro USB на плате. Наличие интерфейса UART-USB позволяет легко подключить плату к компьютеру. NodeMCU может программироваться на языке Си. Существует множество прошивок, дающих возможность писать программы для ESP8266 на языках высокого уровня: Lua, MicroPython, JavaScript, Basic, Лисп [2, 4 – 6]. Кроме того, написаны специальные прошивки для Интернета вещей и домашней автоматизации. NodeMCU часто используется для создания систем умного дома или роботов, управляемых на расстоянии.

Плата из семейства Arduino ориентирована на непрофессиональных пользователей. Программирование встроенного микроконтроллера ATmega328 от Atmel производится на специальном языке Arduino с помощью бесплатной среды программирования Arduino IDE. Uno R3 имеет 32 Кб памяти для программ, 2 Кб ОЗУ и 1 Кб энергонезависимой памяти EEPROM. Программы для Arduino называются скетчами. При загрузке скетча используется Загрузчик Arduino – небольшая программа, заранее загруженная в микроконтроллер на плате. Запрограммировать Arduino Uno R3 можно, подключив плату к компьютеру обычным USB-кабелем [2, 4 – 6] (рис. 1.31).

Официальная отладочная плата от STMicroelectronics с 8-разрядным микроконтроллером STM8S105C6T6 имеет 32 Кб флеш-памяти для программ, 2 Кб оперативной памяти, 1 Кб энергонезависимой памяти EEPROM; пользовательский светодиод и сенсорные кнопки, а также удобную макетную площадку для монтажа дополнительных элементов схемы (рис. 1.32).

Рис. 1.31. Arduino Uno R3

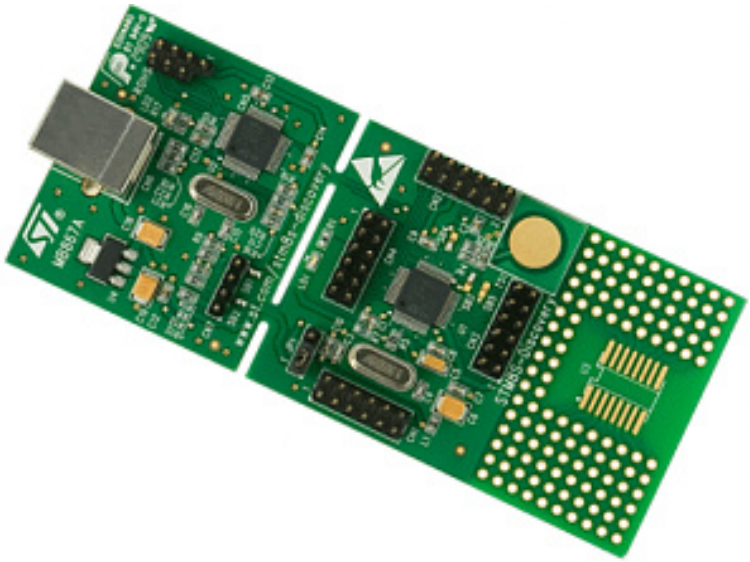

Рис. 1.32. STM8S-Discovery

В плату интегрирован программатор/отладчик ST-Link, позволяющий загружать программы в память микроконтроллера, подключив плату к порту USB. Программатор может быть легко отделен от платы, которая может использоваться в законченном устройстве.

Мощная отладочная плата от ST имеет на борту программатор/отладчик ST-LINK (рис. 1.33) и производительный 32-разрядный ARM-микроконтроллер STM32F100RBT со 128 Кб флеш-памяти и 8 Кб ОЗУ. На плате присутствуют два пользовательских светодиода и кнопка. Эта плата часто рекомендуется в качестве начального средства для знакомства с ARM-микроконтроллерами на ядре Cortex-M3.

Плата от Texas Instruments для быстрого знакомства с 16-разрядными микроконтроллерами MSP430 поставляется с установленным микроконтроллером MSP430G2553 (16 Кб флеш-памяти для программ, 512 байт ОЗУ) и дополнительным MSP430G2452 (рис. 1.34).

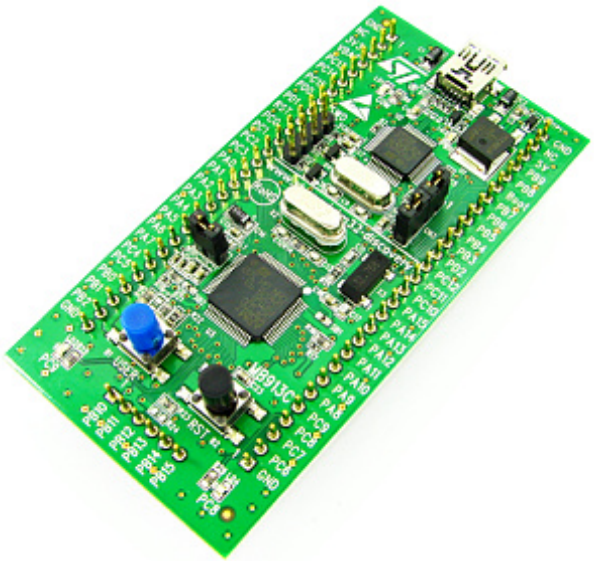

Рис. 1.33. STM32VL-Discovery

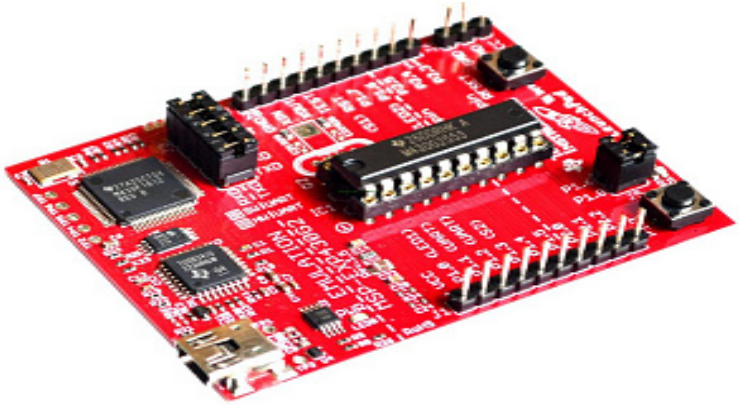

Рис. 1.34. MSP430 LaunchPad

Для составления программ на языке Си можно использовать бесплатную Code Composer Studio от производителя платы. Кроме того, существует бесплатная ардуиноподобная среда Energia [2, 4 – 6].

Глава 2. ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ НА БАЗЕ ПЛАТФОРМЫ ARDUINO

2.1. Описание платформы Arduino

Arduino – это небольшая управляющая плата с собственным процессором и памятью. Помимо них на плате есть два десятка контактов, к которым можно подключать всевозможные компоненты: светодиоды, датчики, моторы, роутеры, сервоприводы и многие другие электронные устройства.

В процессор Arduino можно загрузить программу, которая будет управлять всеми этими устройствами по заданному алгоритму. Таким образом можно создать большое количество уникальных приборов, функционирующих согласно задумке разработчика. На рис. 2.1. приведена плата Arduino Mega, которая является одной из реализаций линейки программируемых платформ Arduino [10, 11].

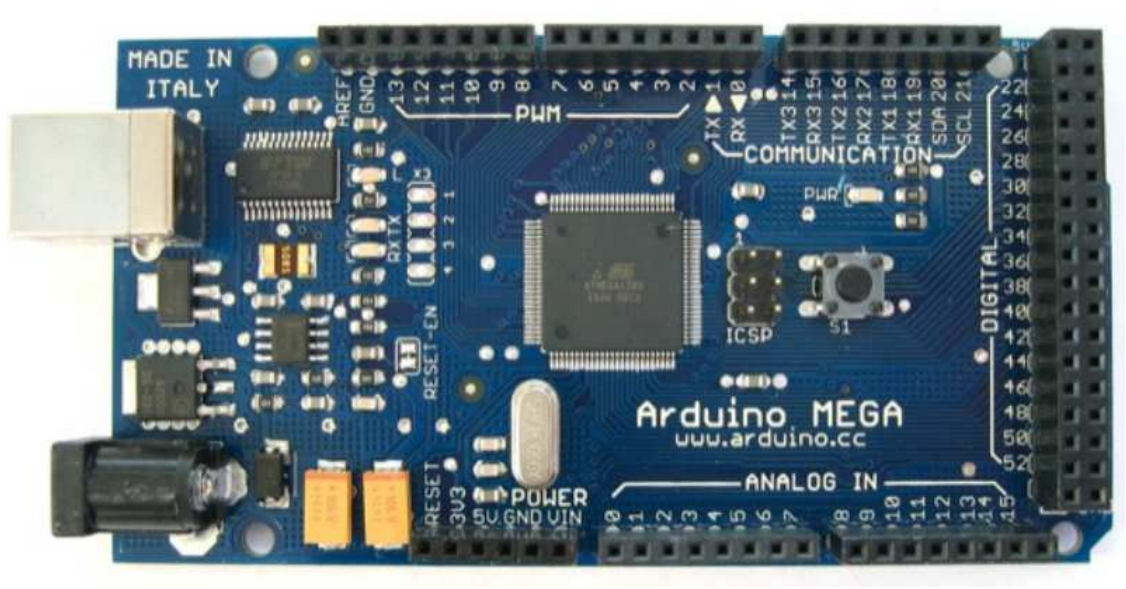

Рис. 2.1. Платформа Arduino Mega

Существует множество микроконтроллеров и платформ для осуществления «physical computing». Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard и многие другие предлагают схожую функциональность. Все эти устройства объединяют разрозненную информацию о программировании и заключают ее в простую в

использовании сборку. Arduino, в свою очередь, тоже упрощает процесс работы с микроконтроллерами, однако имеет ряд преимуществ перед другими устройствами для преподавателей, студентов и любителей:

1. Низкая стоимость – платы Arduino относительно дешевы по сравнению с другими платформами. Самая недорогая версия модуля Arduino может быть собрана вручную, а некоторые даже готовые модули стоят меньше 50 дол.

2. Кросс-платформенность – программное обеспечение Arduino работает под ОС Windows, Macintosh OSX и Linux. Большинство микроконтроллеров ограничивается ОС Windows.

3. Простая и понятная среда программирования – среда Arduino подходит как для начинающих пользователей, так и для опытных. Arduino основана на среде программирования Processing, что очень удобно для преподавателей, так как студенты, работающие с данной средой, будут знакомы и с Arduino [10, 11].

4. Программное обеспечение с возможностью расширения и открытым исходным текстом. ПО Arduino выпускается как инструмент, который может быть дополнен опытными пользователями. Язык может дополняться библиотеками C++. Пользователи, желающие понять технические нюансы, имеют возможность перейти на язык AVR C, на котором основан C++. Соответственно имеется возможность добавить в программу Arduino код из среды AVR-C.

5. Аппаратные средства с возможностью расширения и открытыми принципиальными схемами – микроконтроллеры ATMEGA8 и ATMEGA168 являются основой Arduino. Схемы модулей выпускаются с лицензией Creative Commons, а значит, опытные инженеры имеют возможность создания собственных версий модулей, расширяя и дополняя их. Даже обычные пользователи могут разработать опытные образцы с целью экономии средств и понимания работы [10, 11, 36].

Для программирования Arduino используется упрощенная версия языка C++ с предопределенными функциями. Как и в других Си-подобных языках программирования, есть ряд правил написания кода. Вот самые базовые из них:

- после каждой инструкции необходимо ставить знак точки с запятой (;);

- перед объявлением функции необходимо указать тип данных, возвращаемый функцией, или void, если функция не возвращает значение;
- следует указать тип данных перед объявлением переменной;
- комментарии обозначаются: // Строчный и /* блочный */.

2.2. Цифровые и аналоговые входы/выходы

Цифровые выводы

Выводы платформы Arduino могут работать как входы или как выходы. Данный документ объясняет функционирование выводов в этих режимах. Также необходимо обратить внимание на то, что большинство аналоговых входов Arduino (Atmega) могут конфигурироваться и работать так же, как и цифровые порты ввода/вывода.

Выводы Arduino (Atmega) стандартно настроены как порты ввода, следовательно, не требуется явной декларации в функции pinMode(). Сконфигурированные порты ввода находятся в высокоимпедансном состоянии. Это означает, что порт ввода дает слишком малую нагрузку на схему, в которую он включен. Эквивалентом внутреннему сопротивлению будет резистор сопротивлением 100 МОм, подключенный к выводу микросхемы. Таким образом, для перевода порта ввода из одного состояния в другое требуется маленькое значение тока. Это позволяет применять выводы микросхемы для подключения емкостного датчика касания, фотодиода, аналогового датчика со схемой, похожей на RC-цепь [10, 11].

С другой стороны, если к данному выводу ничего не подключено, то значения на нем будут принимать случайные величины, наводимые электрическими помехами или емкостной взаимосвязью с соседним выводом.

Если на порт ввода не поступает сигнал, то в данном случае рекомендуется задать порту известное состояние. Это делается добавлением подтягивающих резисторов сопротивлением 10 кОм, подключающих вход либо к напряжению +5 В (подтягивающие к питанию резисторы), либо к земле (подтягивающие к земле резисторы).

Микроконтроллер Atmega имеет программируемые встроенные подтягивающие к питанию резисторы 20 кОм. Программирование данных резисторов осуществляется следующим образом.

```
pinMode(pin, INPUT);     // назначить выводу порт ввода
digitalWrite(pin, HIGH);  // включить подтягивающий резистор
```

Подтягивающий резистор пропускает ток, достаточный для того, чтобы слегка светился светодиод, подключенный к выводу, работающему как порт ввода. Также легкое свечение светодиодов означает, что при программировании вывод не был настроен как порт вывода в функции pinMode().

Подтягивающие резисторы управляются теми же регистрами (внутренние адреса памяти микроконтроллера), что управляют состояниями вывода: HIGH или LOW. Следовательно, если вывод работает как порт ввода со значением HIGH (это означает включение подтягивающего к питанию резистора), то конфигурация функцией pinMode() порта вывода на данном выводе микросхемы передаст значение HIGH. Данная процедура работает и в обратном направлении, т. е. если вывод имеет значение HIGH, то конфигурация вывода микросхемы как порта ввода функцией pinMode() включит подтягивающий к питанию резистор [10, 11].

Примечание. Затруднительно использовать 13-й вывод микросхемы Arduino в качестве порта ввода из-за подключенных к нему светодиода и резистора. При подключении подтягивающего к питанию резистора сопротивлением 20 кОм на вводе будет напряжение 1,7 В вместо 5 В, так как происходит падение напряжения на светодиоде и включенном последовательно резисторе. При необходимости использовать вывод 13 микросхемы как цифровой порт ввода требуется подключить между выводом и землей внешний подтягивающий резистор.

Выводы, сконфигурированные как порты вывода, находятся в низкоимпедансном состоянии. Данные выводы могут пропускать через себя достаточно большой ток. Выводы микросхемы Atmega могут быть источником (положительный) или приемником (отрицательный) тока до 40 мА для других устройств. Такого значения тока достаточно, чтобы подключить светодиод (обязателен последовательно включенный резистор), датчики, но недостаточно для большинства реле, соленоидов и двигателей.

Короткие замыкания выводов Arduino или попытки подключить энергоемкие устройства могут повредить выходные транзисторы вывода или весь микроконтроллер Atmega. В большинстве случаев дан-

ные действия приведут к отключению вывода на микроконтроллере, но остальная часть схемы будет работать согласно программе. Рекомендуется к выходам платформы подключать устройства через резисторы сопротивлением 470 Ом или 1 кОм, если устройству не требуется бóльший ток для работы.

Аналоговые входы

Приведем описание портов, работающих как аналоговые входы, платформы Arduino (Atmega8, Atmega168, Atmega328 или Atmega1280).

Микроконтроллеры Atmega, используемые в Arduino, содержат шестиканальный аналого-цифровой преобразователь (АЦП). Разрешение преобразователя составляет 10 бит, что позволяет на выходе получать значения от 0 до 1023. Основным применением аналоговых входов большинства платформ Arduino является чтение аналоговых датчиков, но в то же время они имеют функциональность вводов/выводов широкого применения (GPIO) (то же, что и цифровые порты ввода/вывода 0 – 13) [10, 11].

Таким образом, при необходимости применения дополнительных портов ввода/вывода имеется возможность сконфигурировать неиспользуемые аналоговые входы.

Выводы Arduino, соответствующие аналоговым входам, имеют номера от 14 до 19. Это относится только к выводам Arduino, а не к физическим номерам выводов микроконтроллера Atmega. Аналоговые входы могут использоваться как цифровые выводы портов ввода/вывода. Например, код программы для установки вывода 0 аналогового входа на порт вывода со значением HIGH:

```
pinMode(14, OUTPUT);
digitalWrite(14, HIGH);
```

Выводы аналоговых входов имеют подтягивающие резисторы, работающие как на цифровых выводах. Включение резисторов производится командой

```
digitalWrite(14, HIGH);      // включить резистор на выводе аналогового входа 0, пока вывод работает
                                как порт ввода.
```

Подключение резистора повлияет на величину, сообщаемую функцией analogRead() при использовании некоторых датчиков.

Большинство пользователей используют подтягивающий резистор при применении вывода аналогового входа в его цифровом режиме.

Для вывода, работавшего ранее как цифровой порт вывода, команда analogRead будет работать некорректно. В этом случае рекомендуется сконфигурировать его как аналоговый вход. Аналогично, если вывод работал как цифровой порт вывода со значением HIGH, то обратная установка на ввод подключит подтягивающий резистор.

Руководство на микроконтроллер Atmega не рекомендует производить быстрое переключение между аналоговыми входами для их чтения. Это может вызвать наложение сигналов и внести искажения в аналоговую систему. Однако после работы аналогового входа в цифровом режиме может возникнуть необходимость настроить паузу между чтением функцией analogRead() других входов.

2.3. Широтно-импульсная модуляция и память платформы Arduino

Широтно-импульсная модуляция, или ШИМ, – это операция получения изменяющегося аналогового значения посредством цифровых устройств. Устройства используются для получения прямоугольных импульсов – сигнала, который постоянно переключается между максимальным и минимальным значениями. Данный сигнал моделирует напряжение между максимальным значением (5 В) и минимальным (0 В), изменяя при этом длительность времени включения 5 В относительно включения 0 В. Длительность включения максимального значения называется шириной импульса. Для получения различных аналоговых величин изменяется ширина импульса. Например, при достаточно быстрой смене периодов включения-выключения можно подавать на светодиод постоянный сигнал между 0 и 5 В, тем самым управляя яркостью его свечения [10, 11].

На рис. 2.2 вертикальные линии отмечают постоянные временные периоды. Длительность периода обратно пропорциональна частоте ШИМ, т. е. если частота ШИМ составляет 500 Гц, то зеленые линии будут отмечать интервалы длительностью в 2 мс каждый. Вызов функции analogWrite() с масштабом 0 – 255 означает, что значение analogWrite(255) будет соответствовать 100%-ному рабочему циклу

(постоянное включение 5 В), а значение analogWrite(127) – 50%-ному рабочему циклу.

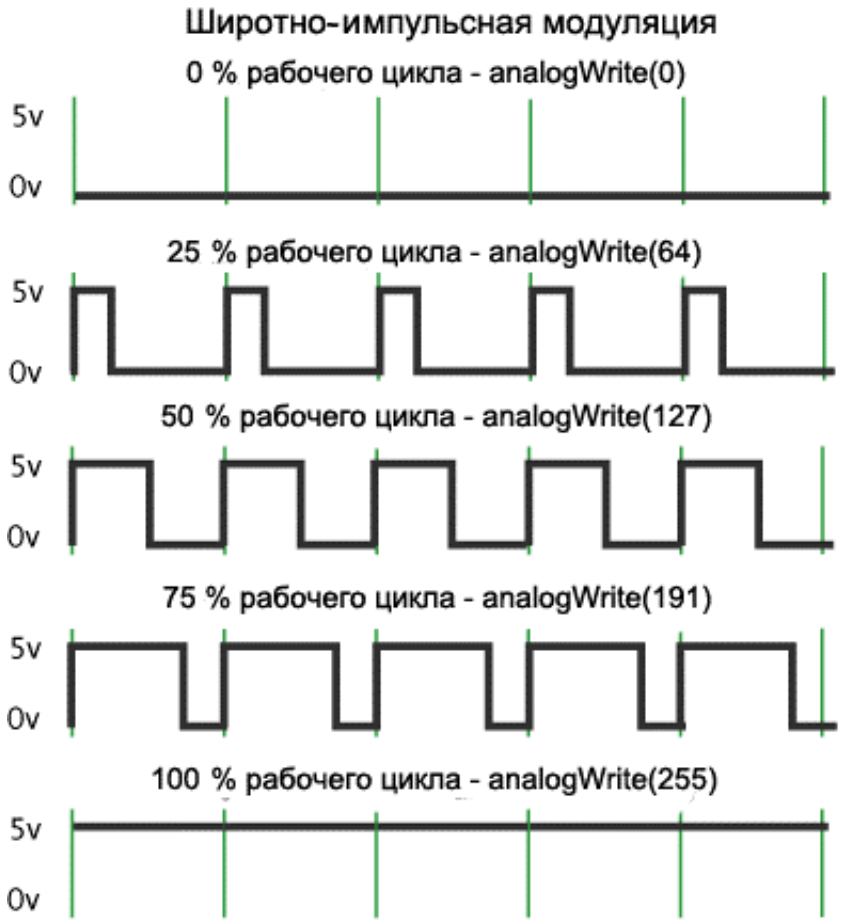

Рис. 2.2. Временные диаграммы широтно-импульсной модуляции

Память в Arduino

В микроконтроллере ATmega168, используемом на платформах Arduino, существует три вида памяти:

- флеш-память – используется для хранения скетчей;
- ОЗУ (статическая оперативная память с произвольным доступом) – используется для хранения и работы переменных.
- EEPROM (энергонезависимая память) – используется для хранения постоянной информации.

Флеш-память и EEPROM являются энергонезависимыми видами памяти (данные сохраняются при отключении питания), ОЗУ – энергозависимой памятью.

Микроконтроллер ATmega168 имеет:

- 16 Кб флеш-памяти (2 Кб используется для хранения загрузчика);
- 1024 байта ОЗУ;
- 512 байт EEPROM.

Необходимо обратить внимание на малый объем ОЗУ, так как большое число строк в скетче может полностью ее израсходовать. Например, следующее объявление:

```
char message[] = "I support the Cape Wind project.";
```

занимает 32 байта из общего объема ОЗУ (каждый знак занимает один байт). При наличии большого объема текста или таблиц для вывода на дисплей возможно полностью использовать допустимые 1024 байта ОЗУ.

При отсутствии свободного места в ОЗУ могут произойти сбои программы, например, она может записаться, но не работать. Для определения данного состояния требуется превратить в комментарии или укоротить строки скетча (без изменения кода). Если после этого программа работает корректно, то на ее выполнение был затрачен весь объем ОЗУ. Существует несколько путей решения данной проблемы:

- при работе скетча с программой на компьютере можно перебросить часть данных или расчетов на компьютер для снижения нагрузки на Arduino;
- при наличии таблиц поиска или других больших массивов можно использовать минимальный тип данных для хранения значений. Например, тип данных int занимает два байта, а byte – только один (но может хранить небольшой диапазон значений);
- неизменяемые строки и данные во время работы скетча можно хранить во флеш-памяти. Для этого необходимо использовать ключ PROGMEM.

Для использования EEPROM необходимо обратиться к библиотеке EEPROM.

2.4. Основные функции при программировании платформы Arduino

Функция Setup

setup()

Функция setup() вызывается, когда стартует программа. Используется для инициализации переменных, определения режимов работы выводов, запуска используемых библиотек и т. д. Функция setup запускается только один раз, после каждой подачи питания или сброса платы Arduino [12].

Пример

```
int buttonPin = 3;
void setup()
{
 Serial.begin(9600);
 pinMode(buttonPin, INPUT);
}
```

Функция Loop

loop()

После вызова функции setup(), которая инициализирует и устанавливает первоначальные значения, функция loop() выполняется раз за разом, позволяя программе совершать вычисления и реагировать на них. Используется для активного управления платой Arduino [Там же].

Пример

```
// в цикле проверяется состояние кнопки,
// и на последовательный порт будет отправлено сообщение,
   если она нажата
void loop()
{
 if (digitalRead(buttonPin) == HIGH)
 serialWrite('H');
 else
 serialWrite('L');
 delay(1000);
}
```

2.5. Основные типы данных для программирования платформы Arduino

Тип boolean

Логический (булевый) тип данных – boolean. Может принимать одно из двух значений true или false. boolean занимает в памяти один байт [12].

Пример

```
int LEDpin = 5;  // Светодиод на входе 5
int switchPin = 13;  // выключатель на порту 13, замыкает на землю
boolean running = false;
void setup()
{
pinMode(LEDpin, OUTPUT);
pinMode(switchPin, INPUT);
digitalWrite(switchPin, HIGH); // включаем подтягивающий резистор
}

void loop()
{
 if (digitalRead(switchPin) == LOW)
 {  // выключатель нажат, так как подтягивающий резистор будет
       давать HIGH на входе, если не замкнут напрямую на землю
 delay(100);   // ждем 0,1с
 running = !running;  // меняем значение булевой переменной
 digitalWrite(LEDpin, running) // включаем или выключаем
                                          светодиод.
 }
}
```

Тип char

Переменная типа char занимает один байт памяти и может хранить один алфавитно-цифровой символ (литеру). При объявлении литеры используются одиночные кавычки: 'A' (двойные кавычки используются при объявлении строки символов – тип string: "ABC") [Там же].

Символ хранится в памяти как число, соответствующее коду символа, в таблице кодировки символов ASCII. Так как символ хранится как число в памяти, над ним возможно производить арифметические действия (например, 'A' + 1 будет 66, так как ASCII код для 'A' – 65).

Тип char знаковый, т. е. число (код), хранящееся в памяти, может принимать значения от –128 до 127. Если необходима беззнаковая однобайтовая переменная, используется тип byte.

Пример

```
char myChar = 'A';
char myChar = 65; // Варианты эквивалентны
```

Тип byte

Тип данных byte – восьмибитное беззнаковое целое число в диапазоне 0…255.

Пример

```
byte c = B10010; // "B" префикс двоичной системы счисления (B10010 = 18 в десятичной системе счисления).
```

Тип int

Тип данных int (от англ. integer – целое число) – один из наиболее часто используемых типов данных для хранения чисел; int занимает 2 байта памяти и может хранить числа от –32 768 до 32 767 (от –2^15 до 2^15 – 1).

Для размещения отрицательных значений int использует так называемый дополнительный код представления числа. Старший бит указывает на отрицательный знак числа, остальные биты инвертируются с добавлением единицы. Arduino компилятор сам заботится о размещении в памяти и представлении отрицательных чисел, поэтому арифметические действия над целыми числами производятся как обычно [12].

Пример

```
int ledPin = 13;
```

Тип unsigned int

Тип данных unsigned int – беззнаковое целое число, так же как и тип int (знаковое), занимает в памяти 2 байта. Но в отличие от int тип

unsigned int может хранить только положительные целые числа в диапазоне от 0 до 65535 (2^16)-1) [12].

Отличие кроется в том, как unsigned int использует старший бит, иногда называемый знаковым битом. Если старший бит равен 1, то для типа int компилятор Arduino считает, что это число отрицательное, а остальные 15 бит несут информацию о модуле целого числа в дополнительном коде представления числа, в то время как unsigned int использует все 16 бит для хранения модуля числа.

Пример

unsigned int ledPin = 13;

Тип long

Описание типа

Тип данных long используется для хранения целых чисел в расширенном диапазоне от –2, 147, 483, 648 до 2, 147, 483, 647. Long занимает 4 байта в памяти [Там же].

Пример

long speedOfLight = 186000L;

Тип float

Тип данных float служит для хранения чисел с плавающей запятой. Этот тип часто используется для операций с данными, считываемыми с аналоговых входов. Диапазон значений от –3.4028235E+38 до 3.4028235E+38. Переменная типа float занимает в памяти 32 бита (4 байта) [Там же].

Тип float имеет точность 6 – 7 знаков, имеются в виду все знаки, а не только мантисса. Обычно для увеличения точности используют другой тип – double, но на платформе Arduino double и float имеют одинаковую точность.

Хранение в памяти чисел с плавающей точкой в двоичной системе обусловливает потерю точности. Так, например, 6.0 / 3.0 не обязательно равен 2.0. Сравнивая два числа с плавающей точкой следует проверять не точное равенство, а разницу между этими числами, меньше ли она некоего выбранного малого порога.

Следует также учитывать, что арифметические операции над числами с плавающей запятой выполняются существенно медленнее, чем над целыми.

Пример

```
float myfloat;
float sensorCalbrate = 1.117;
```

Тип double

Тип данных double, в отличие от большинства языков программирования, имеет ту же точность, что и тип float, и занимает также 4 байта памяти. Тип double поддерживается в Arduino для совместимости кода с другими платформами [12].

Тип string – текстовые строки

Текстовые строки в Arduino объявляются как массив (array) типа char (символов – литер), оканчивающийся символом "конца строки" [Там же].

Ниже приведены варианты объявления и присвоения строк:

```
char Str1[15];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

Массивы

Массивы (arrays) – именованный набор однотипных переменных с доступом к отдельным элементам по их индексу [Там же].

Ниже приведено несколько корректных вариантов объявления массивов:

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

Массив может быть объявлен без непосредственной инициализации элементов массива, как в случае массива myInts.

Массив myPins был объявлен без явного задания размера. Компилятор сам посчитает фактическое количество элементов и создаст в памяти массив необходимого размера.

Размер может быть задан явно, одновременно с инициализацией элементов массива. Обратите внимание, что при создании массива типа char необходим дополнительный элемент массива для нулевого символа.

Тип void

Ключевое слово void используется при объявлении функций, если функция не возвращает никакого значения при ее вызове (в некоторых языках программирования такие функции называют процедурами) [12].

Пример

```
// в функциях "setup" и "loop" производятся некоторые действия,
// но ничего не возвращается во внешнюю программу
void setup()
{
 // ...
}
void loop()
{
 // ...}
```

2.6. Основные операторы программирования платформы Arduino

Управляющие операторы

Оператор if

if (условие) и ==, !=, <, > (операторы сравнения)

if, используется в сочетании с операторами сравнения, проверяет, достигнута ли истинность условия, например, превышает ли входное значение заданное число. Формат оператора if следующий:

```
if (someVariable > 50)
{
// выполнять действия
}
```

Программа проверяет: значение someVariable больше чем 50 или нет. Если да, то выполняются определенные действия. Иначе говоря, если выражение в круглых скобках истинно, выполняются операторы

внутри фигурных скобок. Если нет – программа пропускает этот код [12].

Скобки после оператора if могут быть опущены, в этом случае только следующая строка (обозначенная точкой с запятой) становится оператором, выполняемым в операторе if.

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120){
digitalWrite(LEDpin1, HIGH);
digitalWrite(LEDpin2, HIGH);
}     // все правильно
```

Оператор if..else

Конструкция if..else предоставляет больший контроль над процессом выполнения кода, чем базовый оператор if, позволяя осуществлять несколько проверок, объединенных вместе. Например, аналоговый вход может быть проверен и выполнено одно действие, если на входе меньше 500, или другое действие, если на входе 500 или больше [Там же]. Код при этом может выглядеть так:

```
if (pinFiveInput < 500)
{
 // действие A
}
else
{
 // действие B
}
```

Другой способ создания переходов со взаимоисключающими проверками использует оператор switch case.

Else позволяет делать отличную от указанной в if проверку, чтобы можно было осуществлять сразу несколько взаимоисключающих проверок. Каждая проверка позволяет переходить к следующему за

ней оператору не раньше, чем получит логический результат ИСТИНА. Когда проверка с результатом ИСТИНА найдена, запускается вложенный в нее блок операторов, затем программа игнорирует все следующие строки в конструкции if..else. Если ни одна из проверок не получила результат ИСТИНА, по умолчанию выполняется блок операторов в else, если последний присутствует, и устанавливается действие по умолчанию [12].

Отметим, что конструкция else if может быть использована с или без заключительного else и наоборот. Допускается неограниченное число таких переходов else if.

```
if (pinFiveInput < 500)
{
 // выполнять действие A
}
else if (pinFiveInput >= 1000)
{
 // выполнять действие B
}
else
{
 // выполнять действие C
}
```

Оператор for

Конструкция for используется для повторения блока операторов, заключенных в фигурные скобки. Счетчик приращений обычно используется для приращения и завершения цикла. Оператор for подходит для любых повторяющихся действий и часто используется в сочетании с массивами коллекций данных/выводов [Там же].

Заголовок цикла for состоит из трех частей:

for (initialization; condition; increment) {операторы, выполняющиеся в цикле}

Инициализация (Initialization) выполняется самой первой и один раз. Каждый раз в цикле проверяется условие (condition), если оно верно, выполняются блок операторов и приращение (increment), затем условие проверяется вновь. Когда логическое значение условия становится ложным, цикл завершается.

Пример

```
// Затемнение светодиода с использованием ШИМ-вывода
int PWMpin = 10; // Светодиод последовательно с резистором
                 470 Ом на 10 выводов
void setup()
{
 // настройка не нужна
}

void loop()
{
  for (int i=0; i <= 255; i++){
    analogWrite(PWMpin, i);
    delay(10);
 }
}
```

Цикл for в Си гораздо более гибкий, чем циклы for в других языках программирования, например в Бейсике. Любой из трех или все три элемента заголовка могут быть опущены, хотя точки с запятой обязательны. Также операторы для инициализации, условия и приращения цикла могут быть любыми допустимыми в Си операторами с независимыми переменными и использовать любой тип данных Си, включая данные с плавающей точкой (floats). Эти необычные для цикла for типы операторов позволяют обеспечить программное решение некоторых нестандартных проблем [12].

Например, плавное уменьшение или увеличение уровня сигнала на светодиод с помощью одного цикла for:

```
void loop()
{
 int x = 1;
 for (int i = 0; i > -1; i = i + x){
   analogWrite(PWMpin, i);
   if (i == 255) x = -1;  // переключение управления на максимуме
   delay(10);
 }
}
```

Оператор Switch

Подобно конструкции if, switch...case управляет процессом выполнения программы, позволяя программисту задавать альтернативный код, который будет выполняться при разных условиях. В частности, оператор switch сравнивает значение переменной со значением, определенным в операторах case. Когда найден оператор case, значение которого равно значению переменной, в этом операторе выполняется программный код [12].

Ключевое слово break является командой выхода из оператора case и обычно используется в конце каждого case. Без оператора break оператор switch будет продолжать вычислять следующие выражения, пока не достигнет break или конца оператора switch.

Пример

```
switch (var) {
   case 1:
     // выполняется, когда var равно 1
     break;
   case 2:
    // выполняется, когда var равно 2
    break;
   default:
    // выполняется, если не выбрана ни одна альтернатива
    // default необязателен
 }
```

Циклы while

While будет вычислять в цикле непрерывно и бесконечно до тех пор, пока выражение в круглых скобках () не станет равно логическому ЛОЖНО. Что-то должно изменять значение проверяемой переменной, иначе выход из цикла while никогда не будет достигнут. Это изменение может происходить как в программном коде, например при увеличении переменной, так и во внешних условиях, например при тестировании датчика [Там же].

Пример

```
var = 0;

while(var < 200){
 // выполнить что-то, повторив 200 раз
 var++;
}
```

Оператор break

Break используется для принудительного выхода из циклов do, for или while, не дожидаясь завершения цикла по условию. Он также используется для выхода из оператора switch [12].

Пример

```
for (x = 0; x < 255; x ++)
  {
  digitalWrite(PWMpin, x);
  sens = analogRead(sensorPin);
  if (sens > threshold){    // выходим из цикла, если есть сигнал
                                с датчика
    x = 0;
    break;
    }
  delay(50);
  }
```

Оператор continue

Оператор continue пропускает оставшиеся операторы в текущем шаге цикла. Вместо них выполняется проверка условного выражения цикла, которая происходит при каждой следующей итерации [Там же].

Пример

```
for (x = 0; x < 255; x ++)
{
  if (x > 40 && x < 120){  // если истина, то прыгаем сразу на сле-
                                дующую итерацию цикла
    continue;
  }

  digitalWrite(PWMpin, x);
  delay(50);

}
```

Оператор return

Прекращает вычисления в функции и возвращает значение из прерванной функции в вызывающую, если это нужно [Там же].

Примеры:

Функция сравнивает значение на датчике входа с пороговым

```
int checkSensor(){
  if (analogRead(0) > 400) {
    return 1;
    else{
      return 0;
    }
  }
```

С помощью ключевого слова return удобно тестировать блоки кода без «закомментирования» больших кусков с возможным ошибочным кодом.

```
void loop(){
  // здесь блестящая идея тестирования кода
  return;
  // оставшаяся часть неправильно функционирующего варианта
     здесь
  // этот код никогда не будет выполняться
}
```

Оператор goto

Условное «перемещение» выполнения программы к определенной метке-указателю в самой программе, при этом пропускается весь код до самой метки, а исполняется – после нее [12].

Использование goto не рекомендуется в С программировании, многие авторы не советуют применять его вообще, так как это не является необходимым (с их точки зрения). Причины такого мнения заключаются в том, что программист при частом использовании в коде команды goto может запустить программу в бесконечный цикл, который потом трудно будет найти – отладка программы значительно усложнится. С другой стороны, если взглянуть на ассемблерный код, то там часто используется подобный переход по метке.

При разумном применении команда может значительно упростить код программы и сохранить время программиста, например, в случае необходимости выхода из глубоких циклов for, while, проверок if и прочих многократно вложенных конструкций.

Пример

```
for(byte r = 0; r < 255; r++){
   for(byte g = 255; g > -1; g--){
      for(byte b = 0; b < 255; b++){
         if (analogRead(0) > 250){ goto bailout;}
         // еще код
      }
   }
}
bailout:
```

2.7. Прикладные функции для программирования платформы Arduino

Цифровой ввод/вывод

Функция pinMode

Устанавливает режим работы заданного входа/выхода (pin) как входа или как выхода.

Синтаксис

pinMode(pin, mode)

Параметры

- pin: номер входа/выхода (pin), который Вы хотите установить
- mode: режим одно из двух значений – INPUT или OUTPUT – устанавливает на вход или выход соответственно.

Возвращаемое значение

нет

Пример

```
 int ledPin = 13;   // Светодиод, подключенный к входу/выходу 13
void setup()
{
 pinMode(ledPin, OUTPUT);     // устанавливает режим работы – выход
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // включает светодиод
```

```
  delay(1000);   // ждет секунду
  digitalWrite(ledPin, LOW);   // выключает светодиод
  delay(1000);   // ждет секунду
}
```

Аналоговые входы (analog pins) могут быть использованы как цифровые входы/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5) [12].

Функция digitalWrite()

Подает HIGH или LOW значение на цифровой вход/выход (pin).

Если вход/выход (pin) был установлен в режим выход (OUTPUT) функцией pinMode(), то для значения HIGH напряжение на соответствующем входе/выходе (pin) будет 5 В (3,3 В для 3,3 В плат) и 0 В(земля) для LOW.

Если вход/выход (pin) был установлен в режим вход (INPUT), то функция digitalWrite со значением HIGH будет активировать внутренний 20K нагрузочный резистор. Подача LOW, в свою очередь, отключает этот резистор. Нагрузочного резистора достаточно, чтобы светодиод, подключенный к входу, светил тускло. Если вдруг светодиод работает, но очень тускло, вероятно, необходимо установить режим выход (OUTPUT) функцией pinMode() [Там же].

Замечание. Вход/выход 13 сложнее использовать как цифровой вход, так как он имеет встроенные в плату резистор и светодиод. Если вы активируете еще внутренний нагрузочный резистор 20 K, то напряжение на этом входе будет около 1,7 В вместо ожидаемых 5 В, так как светодиод и добавочный резистор снижают напряжение, т. е. вы всегда будете получать LOW. Если же вам необходимо использовать 13-й вход/выход, то используйте внешний нагрузочный резистор [Там же].

Синтаксис

digitalWrite(pin, value)

Параметры

- pin: номер вход/выхода(pin)
- value: значение HIGH или LOW

Возвращаемое значение

нет

Пример

```
int ledPin = 13;   // Светодиод, подключенный к входу/выводу 13

void setup()
{
  pinMode(ledPin, OUTPUT);    // устанавливаем режим работы – выход
}
 }

void loop()
{
  digitalWrite(ledPin, HIGH);   // включает светодиод
  delay(1000);   // ждет секунду
  digitalWrite(ledPin, LOW);   // выключает светодиод
  delay(1000);   // ждет секунду
}
```

Аналоговые входы (analog pins) могут быть использованы как цифровые входы/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5).

Функция digitalRead()

Функция считывает значение с заданного входа HIGH или LOW.

Синтаксис

digitalRead(pin)

Параметры

pin: номер входа/выхода (pin), который Вы хотите считать

Возвращаемое значение

HIGH или LOW

Пример

```
  int ledPin = 13;  // Светодиод, подключенный к входу/выходу 13
int inPin = 7;   // кнопка на входе 7
int val = 0;   // переменная для хранения значения
```

```
void setup()
{
 pinMode(ledPin, OUTPUT); // устанавливает режим работы –
                          выход для 13-го входа/выхода (pin)
 pinMode(inPin, INPUT);   // устанавливает режим работы – вход
                          для 7-го входа/выхода (pin)
}

void loop()
{
 val = digitalRead(inPin);   // считываем значение с входа
 digitalWrite(ledPin, val);  // устанавливаем значение на светодио-
                             де, равное значению входа кнопки
}
```

Если вход не подключен, то digitalRead может возвращать значения HIGH или LOW случайным образом.

Аналоговые входы (analog pins) могут быть использованы как цифровые входы/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5).

Аналоговый ввод/вывод

Функция analogRead()

Функция считывает значение с указанного аналогового входа. Большинство плат Arduino имеют 6 каналов (8 каналов у платы Mini и Nano, 16 – у Mega) с 10-битным аналого-цифровым преобразователем. Напряжение, поданное на аналоговый вход (обычно от 0 до 5 В), будет преобразовано в значение от 0 до 1023, это 1024 шага с разрешением 0,0049 В. Разброс напряжения и шаг могут быть изменены функцией analogReference() [12].

Считывание значения с аналогового входа занимает примерно 100 мкс (0.0001 с), т. е. максимальная частота считывания приблизительно 10,000 раз в секунду.

Синтаксис

analogRead(pin)

Параметры

pin: номер порта аналогового входа, с которого будет производиться считывание (A0...A5 для большинства плат, 0...7 для Mini и Nano и 0…15 для Mega).

Если аналоговый вход не подключен, то значения, возвращаемые функцией analogRead(), могут принимать случайные величины.

Пример

```
 int analogPin = 3; // номер порта, к которому подключен потенциометр
int val = 0;  // переменная для хранения считываемого значения

void setup()
{
 Serial.begin(9600);  // установка связи по serial
}

void loop()
{
 val = analogRead(analogPin); // считываем значение
 Serial.println(val);  // выводим полученное значение
}
```

Функция analogReference()

Функция определяет опорное напряжение, относительно которого происходят аналоговые измерения. Функция analogRead() возвращает значение с разрешением 10 бит пропорционально входному напряжению на аналоговом входе и в зависимости от опорного напряжения [12].

Возможные настройки:

- DEFAULT: стандартное опорное напряжение 5 В (на платформах с напряжением питания 5 В) или 3,3 В (на платформах с напряжением питания 3,3 В).
- INTERNAL: встроенное опорное напряжение 1,1 В на микроконтроллерах ATmega168 и ATmega328 и 2,56 В на ATmega8.
- INTERNAL1V1: встроенное опорное напряжение 1,1 В (Arduino Mega).

• INTERNAL2V56: встроенное опорное напряжение 2,56 В (Arduino Mega).

• EXTERNAL: внешний источник опорного напряжения, подключенный к выводу AREF.

Синтаксис

analogReference(type)

Параметры

type: определяет используемое опорное напряжение (DEFAULT, INTERNAL или EXTERNAL).

Возвращаемое значение

нет

Внешнее напряжение рекомендуется подключать к выводу AREF через резистор сопротивлением 5 кОм. Таким образом уменьшается риск повреждения микросхемы Atmega, если настройки analogReference не совпадают с возможностями платформы. Однако при этом произойдет небольшая просадка напряжения вследствие того, что имеется встроенный резистор на 32 кОм, подключенный к выводу AREF. В этом случае оба резистора работают как делитель напряжения. Подсоединение внешнего резистора позволяет быстро переключаться на напряжение 3,3 В вывода AREF с напряжения 5 В DEFAULT без конфигурации аппаратной части и АЦП.

Использование вывода AREF

Напряжение, подключенное к выводу AREF, конвертируется АЦП и затем определяется значение напряжения, при котором АЦП выдает самое высокое цифровое значение, т. е. 1023. Другие значения напряжения, поступающие в АЦП, конвертируются пропорционально. Таким образом, при настройке DEFAULT 5 В значение напряжения 2,5 В в АЦП будет конвертироваться в 512 [12].

В стандартной конфигурации платформ Arduino вывод AREF (вывод 21 Atmega) не задействован. В этом случае при настройке DEFAULT к выводу подключается внутреннее напряжение AVCC. Соединение является низкоимпедансным и любое напряжение, подведенное к выводу в этот момент, может повредить микросхему ATMEGA.

Настройкой INTERNAL к выводу AREF подключается внутреннее напряжение 1,1 В (или 2,56 микросхемы ATmega8). При этом напряжение, соответствующее или превышающее 1,1 В, будет кон-

вертироваться АЦП в 1023. Другие значения напряжения конвертируются пропорционально.

Внутреннее подключение источника 1,1 В к выводу является высокоимпедансным и означает, что измерение напряжения на выводе может быть произведено только мультиметром с высоким сопротивлением. Ошибочное подключение напряжения к выводу AREF при этой настройке функции analogReference не повредит микросхему, но превысит значение 1,1 В. В этом случае АЦП будет конвертировать напряжение внешнего источника. Во избежание вышеописанных проблем рекомендуется подключать внешнее напряжение через резистор 5 кОм.

Рекомендуемой настройкой для вывода AREF служит EXTERNAL. При этом происходит отключение обоих внутренних источников, и внешнее напряжение будет опорным для АЦП.

Функция analogWrite()

Выдает аналоговую величину (ШИМ волну) на порт входа/выхода. Функция может быть полезна для управления яркостью подключенного светодиода или скоростью электродвигателя. После вызова analogWrite() на выходе будет генерироваться постоянная прямоугольная волна с заданной шириной импульса до следующего вызова analogWrite (или вызова digitalWrite, или digitalRead на том же порту входа/выхода). Частота ШИМ сигнала приблизительно 490 Гц [12].

На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11, на плате Arduino Mega – порты со 2-го по 13-й. На более ранних версиях плат Arduino analogWrite() работал только на портах 9, 10 и 11.

Для вызова analogWrite() нет необходимости устанавливать тип входа/выхода функцией pinMode().

Функция analogWrite никак не связана с аналоговыми входами и функцией analogRead.

Синтаксис

analogWrite(pin, value)

Параметры

- pin: порт входа/выхода, на который подаем ШИМ сигнал.
- value: период рабочего цикла – значение между 0 (полностью выключено) и 255 (сигнал подан постоянно).

Возвращаемое значение

нет

Замечание

Период ШИМ сигнала на портах входа/выхода 5 и 6 будет несколько длиннее. Это связано с тем, что таймер для данных выходов также задействован функциями millis() и delay(). Данный эффект более заметен при установке коротких периодов ШИМ сигнала (0 – 10).

Пример

Задание яркости светодиода пропорционально значению, снимаемому с потенциометра.

```
 int ledPin = 9;    // светодиод подключен к выходу 9
int analogPin = 3;    // потенциометр подключен к выходу 3
int val = 0;    // переменная для хранения значения

void setup()
{
  pinMode(ledPin, OUTPUT);   // установка порта на выход
}

void loop()
{
 val = analogRead(analogPin); // считываем значение с порта, подключенного к потенциометру
 analogWrite(ledPin, val / 4); // analogRead возвращает значения от 0 до 1023, analogWrite должно быть в диапазоне от 0 до 255
}
```

Работа со временем

Функция millis()

Возвращает количество миллисекунд с момента начала выполнения текущей программы на плате Arduino. Это количество сбрасывается на ноль вследствие переполнения значения приблизительно через 50 дней [12].

Параметры: нет

Возвращаемое значение: Количество миллисекунд с момента начала выполнения программы (unsigned long).

Пример

```
unsigned long time;
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Time: ");
  time = millis();
 // выводит количество миллисекунд с момента начала выполнения программы
  Serial.println(time);
  // ждет секунду перед следующей итерацией цикла.
  delay(1000);
}
```

Функция micros()

Возвращает количество микросекунд с момента начала выполнения текущей программы на плате Arduino. Значение переполняется и сбрасывается на ноль приблизительно через 70 мин. На 16 МГц платах Arduino (Duemilanove и Nano) функция micros() имеет разрешение 4 мкс (возвращаемое значение всегда кратно 4). На 8 МГц платах (Arduino Lilypad) разрешение функции 8 мкс (в одной секунде 1 000 мс и 1 000 000 мкс) [12].

Параметры: нет

Возвращаемое значение: Количество микросекунд с момента начала выполнения программы (unsigned long).

Пример

```
unsigned long time;
void setup(){
  Serial.begin(9600);
}

void loop(){
  Serial.print("Time: ");
```

```
  time = micros();
  // выводит количество микросекунд с момента начала выполнения программы
  Serial.println(time);
  // ждет секунду перед следующей итерацией цикла.
  delay(1000);
}
```

Функция delay()

Останавливает выполнение программы на заданное в параметре количество миллисекунд (1000 мс в 1 с) [12].

Синтаксис

delay(ms)

Параметры

ms: количество миллисекунд, на которое приостанавливается выполнение программы (unsigned long).

Возвращаемое значение

Нет

Пример

```
int ledPin = 13;   // светодиод подключен на порт 13

void setup()
{
 pinMode(ledPin, OUTPUT);   // устанавливается режим порта – выход
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // включаем светодиод
  delay(1000);   // ожидаем секунду
  digitalWrite(ledPin, LOW);   // выключаем светодиод
  delay(1000);   // ожидаем секунду
}
```

Не рекомендуется использовать эту функцию для событий длиннее 10 мс, так как во время останова не могут быть произведены манипуляции с портом, не могут быть считаны сенсоры или произве-

дены математические операции. В качестве альтернативного подхода возможно контролирование времени выполнения тех или иных функций с помощью millis().

Активность платы останавливается функцией delay(). Тем не менее работа прерываний не останавливается, продолжается запись последовательно (serial) передаваемых данных на RX порт, ШИМ сигнал (analogWrite) продолжает генерироваться на портах.

Функция delayMicroseconds()

Останавливает выполнение программы на заданное в параметре количество микросекунд (1 000 000 мкс в 1 с) [12].

В данной версии Arduino максимальная пауза, воспроизводимая корректно, – 16383. Возможно, это будет изменено в следующих версиях Arduino. Для остановки выполнения программы более чем на несколько тысяч микросекунд рекомендуется использовать функцию delay().

Синтаксис

delayMicroseconds(us)

Параметры

us: количество микросекунд, на которое приостанавливается выполнение программы. (unsigned int)

Возвращаемое значение: Нет

Пример

```
int outPin = 8;   // цифровой порт входа/выхода 8
void setup()
{
 pinMode(outPin, OUTPUT);     // устанавливается режим порта –
                                 выход
}

void loop()
{
  digitalWrite(outPin, HIGH);    // подаем HIGH на выход
  delayMicroseconds(50);    // ожидаем 50 микросекунд
  digitalWrite(outPin, LOW);    // устанавливаем LOW на выходе
  delayMicroseconds(50);    // ожидаем 50 микросекунд
}
```

Генераторы случайных значений

Функция randomSeed(seed)

Функция randomSeed() инициализирует генератор псевдослучайных чисел. Генерируемая последовательность случайных чисел очень длинная и всегда одна и та же. Точка в этой последовательности, с которой начинается генерация чисел, зависит от параметра seed [12].

Если при каждом запуске программы необходимо получать разные последовательности значений, генерируемых функцией random(), то необходимо инициализировать генератор псевдослучайных чисел со случайным параметром. Например, можно использовать значение, отдаваемое функцией analogRead() с неподключенного порта входа/выхода.

В некоторых случаях необходимо получать одинаковую последовательность при каждом запуске программы на Arduino. В этом случае инициализировать генератор псевдослучайных чисел следует вызовом функции randomSeed() с фиксированным параметром.

Параметры

• seed: параметр, задающий начало выдачи псевдослучайных значений на последовательности (см. выше) (int, long).

Возвращаемое значение

нет

Пример

```
long randNumber;

void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

void loop(){
  randNumber = random(300);
  Serial.println(randNumber);

  delay(50);
}
```

Функция random()

Функция random() возвращает псевдослучайное число.

Синтаксис

random(max)

random(min, max)

Параметры

- min: нижняя граница случайных значений включительно (опционально).
- max: верхняя граница случайных значений не включительно.

Возвращаемое значение

Случайное число между min и max-1 (long).

Если при каждом запуске программы необходимо получать разные последовательности значений, генерируемых функцией random(), то необходимо инициализировать генератор псевдослучайных чисел со случайным параметром. Например, можно использовать значение, отдаваемое функцией analogRead() с неподключенного порта входа/выхода [12].

В некоторых случаях необходимо получать одинаковую последовательность при каждом запуске программы на Arduino. В этом случае инициализировать генератор псевдослучайных чисел следует вызовом функции randomSeed() с фиксированным параметром.

Пример

```
long randNumber;

void setup(){
  Serial.begin(9600);

// если порт 0 не подключен, то генератор псевдослучайных чисел
// будет инициализироваться функцией randomSeed() со случайного
// значения при каждом запуске программы из-за «шума» на порту
 randomSeed(analogRead(0));
}

void loop() {
  // выводим случайное число из диапазона 0...299
  randNumber = random(300);
  Serial.println(randNumber);
  // выводим случайное число из диапазона 10…19
```

```
  randNumber = random(10, 20);
  Serial.println(randNumber);

  delay(50);
}
```

2.8. Аппаратные прерывания платформы Arduino

Часто при работе с проектами на микроконтроллерах требуется запускать фоновую функцию через равные промежутки времени. Это реализуется установкой аппаратного таймера для выработки прерывания, которое запускает программу обработки прерываний (Interrupt Service Routine, ISR) для управления периодическим прерыванием.

Микроконтроллер ATMega168 имеет три внутренних аппаратных таймера. Хотя библиотека Arduino позволяет использовать некоторые свойства таймеров, нельзя напрямую применять таймер для выработки периодических прерываний [10].

Как видно из названия, прерывания – это сигналы, прерывающие нормальное течение программы. Прерывания обычно используются для аппаратных устройств, требующих немедленной реакции на появление событий. Например, система последовательного порта или UART (универсальный асинхронный приемопередатчик) микроконтроллера должны быть обслужены при получении нового символа. Если этого не сделать быстро, новый символ может быть потерян.

При поступлении нового символа UART генерирует прерывание. Микроконтроллер останавливает выполнение основной программы (вашего приложения) и перескакивает на программу обработки прерываний (ISR), предназначенную для данного прерывания, – это прерывание по полученному символу. Программа ISR захватывает новый символ из UART, помещает в буфер, затем очищает прерывание и выполняет возврат. Когда ISR выполняет возврат, микроконтроллер возвращается в основную программу и продолжает ее с точки вызова. Все это происходит в фоновом режиме и не влияет напрямую на основной код вашего приложения [Там же].

Если запускается много прерываний или прерывания генерирует быстродействующий таймер, основная программа будет выполняться медленнее, так как микроконтроллер распределяет свое машинное время между основной программой и всеми функциями обработки прерываний.

Теоретически время от времени можно просто проверять новый символ вместо использования такого сложного процесса прерывания. Однако это не является оптимальным методом. Например, пусть существует последовательный порт со скоростью передачи данных 9600 бод. Это означает, что каждый бит символа посылается с частотой 9600 Гц или около 10 кГц. На каждый бит уходит 100 мкс. Около 10 бит требуется, чтобы послать один символ, так что один полный символ будет приниматься каждую миллисекунду. Если UART буферизован, необходимо извлечь последний символ до завершения приема следующего, что займет время, равное 1 мс. Если UART не буферизован, то необходимо избавиться от символа за 1 бит или 1 мкс. Рассмотрим для начала буферизованный пример [10].

Необходимо проверять получение байта быстрее, чем каждую миллисекунду, чтобы предотвратить потерю данных. Применительно к Arduino это означает, что функция цикла должна обращаться для чтения статуса UART и, возможно, байта данных 1000 раз в секунду. Это легко выполнимо, но сильно усложнит код, который нужно написать. До тех пор, пока функция цикла не требует больше 1 мс до завершения, это не вызовет проблем. Но если необходимо обслуживать несколько устройств ввода/вывода, то придется работать на гораздо большей скорости передачи.

С прерываниями не нужно отслеживать поступление символа. Аппаратура подает сигнал с помощью прерывания, и процессор быстро вызовет ISR, чтобы вовремя захватить символ. Вместо выделения огромной доли процессорного времени на проверку статуса UART можно устанавливать аппаратное прерывание и выполнять необходимые действия в ISR. При этом основная программа напрямую не затрагивается и от аппаратного устройства не требуется особых возможностей [Там же].

Прерывание по таймеру

Ниже приведен пример использования программного таймера 2 для периодических прерываний. Например, задача заключается в генерации частоты биений в звуковых проектах Arduino. Для того чтобы выводить тон или частоту, необходимо переключать порт ввода/вывода на согласованной частоте. Это можно делать с использованием циклов задержки. В этом случае процессор будет занят, ничего не выполняя, но ожидая точного времени переключения вывода. С использованием прерывания по таймеру можно решать другие задачи, а вывод будет переключаться с помощью ISR, когда таймер подаст сигнал, что время пришло.

Для начала необходимо установить таймер для подачи сигнала с прерыванием в нужное время. Вместо прокрутки бесполезного цикла для задержки по времени главная программа может выполнять что-то другое, например, контролировать датчик движения или управлять электроприводом. Какую бы задачу не выполняла основная программа, процессорное время для получения задержек не потребуется [10].

Таймеры на Arduino

Таймер 0 (системное время, ШИМ 5 и 6) применяется для хранения счетчика времени работы программы. Функция millis() возвращает число миллисекунд с момента запуска программы, используя ISR глобального приращения таймера 0. Таймер 0 также реализует ШИМ на выводах 5 и 6.

Таймер 1 (ШИМ 9 и 10) используется для реализации ШИМ для цифровых выводов 9 и 10; таймер 2 (ШИМ 3 и 11) – для управления выходами ШИМ для цифровых выводов 3 и 11.

Таймер 0 имеет назначенную таймеру процедуру ISR. Это означает, что можно использовать таймер 1 и/или таймер 2 под задачи разработчика, но при этом ШИМ на некоторых портах ввода/вывода станет недоступна [1].

Приведенная в листинге 2.1 программа показывает созданную функцию установки таймера 2. Эта функция подключает прерывание по переполнению таймера 2, устанавливает предварительно заданный масштаб для таймера, подсчитывает его загружаемое значение, дающее желаемую частоту прерываний по времени.

Листинг 2.1

```
#define TIMER_CLOCK_FREQ 2000000.0 //2MHz for /8 prescale from 16MHz

// Установка Таймера2.
// Конфигурирует 8-битный Таймер2 ATMega168 для выработки прерывания
// с заданной частотой.
// Возвращает начальное значение таймера, которое должно быть загружено в TCNT2
// внутри вашей процедуры ISR.
// Смотри пример использования ниже.
unsigned char SetupTimer2(float timeoutFrequency){
unsigned char result; //Начальное значение таймера.

// Подсчет начального значения таймера
result=(int)((257.0-(TIMER_CLOCK_FREQ/timeoutFrequency))+0.5);
// 257 на самом деле должно быть 256, но я получил лучшие результаты с 257.

// Установки Таймер2: Делитель частоты /8, режим 0
// Частота = 16 МГц/8 = 2 МГц или 0,5 мкс
// Делитель /8 дает хороший рабочий диапазон
// так что сейчас мы просто жестко запрограммируем это.
TCCR2A = 0;
TCCR2B = 0<<CS22 | 1<<CS21 | 0<<CS20;

// Подключение прерывания по переполнению таймера 2
TIMSK2 = 1<<TOIE2;

// загружает таймер для первого цикла
TCNT2=result;

return(result);
}
```

Сначала определяют тактовую частоту таймера. Показано, что установлена тактовая частота 2 МГц, так как используется деление на 8 (делитель частоты) опорной частоты 16 МГц. Это определение улучшает внешний вид программы и может быть полезно в некоторых приложениях.

Функция имеет один аргумент – желаемую частоту прерываний – и возвращает значение, которое необходимо перезагружать в таймер в процедуре ISR. Функция не ограничивает требуемую частоту, но не следует слишком ее завышать.

Далее подсчитывают значение, перезагружаемое в таймер. Это очень простой подсчет, но требует выполнения операций с плавающей точкой. Плюсом будет то, что это необходимо сделать только один раз, поскольку операции с плавающей точкой требуют большого количества машинного времени. Примем, что таймер будет установлен на 2 МГц при каждом счете. Загружаемое значение – это число отсчетов, которое необходимо произвести при частоте 2 МГц между прерываниями [10].

Следующий участок кода устанавливает таймер в режим 0 и выбирает делитель частоты /8. Режим 0 – это базовый режим таймера, а делитель /8 показывает, что используется частота 2 МГц или 0,5 мкс на отсчет.

Далее подключают прерывание по переполнению. После выполнения этого кода микроконтроллер будет вызывать ISR каждый раз, когда счетчик прокрутится от 0xFF до 0×00. Это случится, когда счетчик просчитает от загруженного значения через FF и назад до 00. Далее значение счетчика загружаем в таймер и возвращаем загруженное значение, чтобы ISR могла использовать его позже.

Загрузка микроконтроллера прерываниями

Для того чтобы дать общее представление о работе таймера, предположим, что таймер ISR запускался бы каждые 20 мкс. Процессор, работающий на 16 МГц, может выполнить одну машинную команду каждые 63 нс или около 320 машинных команд для каждого цикла прерывания (20 мкс). Предположим также, что исполнение каждой строки программы на C++ может занять много машинных команд. Каждая инструкция, используемая в ISR, отнимает время, доступное для исполнения любой другой программы. Если бы ISR использовала около 150 машинных циклов, половина доступного про-

цессорного времени была бы израсходована. При активных прерываниях главная программа откладывалась бы около ½ времени, занимаемого ей в других случаях. 150 машинных команд – не очень большая программа на C++, поэтому необходимо уделять этому моменту особое внимание.

Если программа ISR будет слишком большой по времени, главная программа будет выполняться крайне медленно, если же ISR будет длиннее, чем продолжительность цикла таймера, то практически никогда не выполнится основная программа, и в конце концов произойдет сбой системного стека [10, 11].

Измерение загрузки прерываниями

В листинге 2.1 таймер не установлен в режим, когда он перезагружается автоматически. Это значит, что ISR должна перезагрузить таймер для следующего интервала счета. Было бы точнее иметь автоматически перезагружаемый таймер, но, используя этот режим, можно измерить время, проводимое в ISR, и соответственно исправить время, загружаемое в таймер. При помощи этой коррекции можно получить время выполнения ISR.

В листинге 2.2 показана программа ISR для таймера 2

Листинг 2.2

```
#define TOGGLE_IO 9   // вывод Arduino для переключения по таймеру ISR
// Таймер 2 указатель вектора прерывания по переполнению
ISR(TIMER2_OVF_vect) {
// Переключение IO-вывода в другое состояние.
digitalWrite(TOGGLE_IO,!digitalRead(TOGGLE_IO));
// Захват текущего значения таймера. Это величина ошибки
// из-за задержки обработки прерывания и работы этой функции
latency=TCNT2;
// Перезагрузка таймера и коррекция по задержке
TCNT2=latency+timerLoadValue;
}
```

Эта функция короткая и ее основная задача – переключать порт ввода/вывода. После переключения она захватывает текущее значение таймера и использует его для коррекции задержки на перезагрузку таймера. Значение задержки глобальное, его может отслеживать

главная программа для вышеупомянутых измерений загрузки. Это число тактов на частоте 2 МГц, которое занимает эта ISR для выполнения своих функций.

Необходимо помнить, что ISR должна быть короткой, так как она вызывается каждые 20 мкс при таймере, настроенном на 50 кГц. Тестирование программы показало среднюю задержку около 20 тактов, что составляет до 45 % загрузки процессора, т. е. из-за ISR главная программа будет выполняться в среднем примерно на 45 % медленнее. ISR была бы гораздо быстрее, если использовать прямое обращение к выводу с помощью регистров порта ввода/вывода.

Главная программа. Функция Setup()

Функция Setup(), приведенная в листинге 2.3, вызывается с помощью системной программы Arduino однократно при ее запуске. Она инициализирует порты ввода/вывода и таймер. Она также выполняет вывод в последовательный порт, показывающий, что программа запущена.

Листинг 2.3

```
void setup(void) {
// Устанавливает порт, который нам нужно переключать в ISR, выходным.
pinMode(TOGGLE_IO,OUTPUT);
// Запускает последовательный порт
Serial.begin(9600);
// Сообщение о запуске программы
Serial.println("Timer2 Test");
// Запускает таймер и получает загружаемое значение таймера.
timerLoadValue=SetupTimer2(44100);
// Выводит загружаемое значение таймера
Serial.print("Timer2 Load:");
Serial.println(timerLoadValue,HEX);
}
```

Setup() начинается с установки переключаемого порта в режим выходного порта для использования его в ISR. Затем она активирует последовательный порт и выводит текстовое сообщение, чтобы показать, что программа работает.

Далее вызывается функция SetupTimer2 с установленной частотой 44100 Гц и общей частотой дискретизации звука. Возвращаемое значение сохраняется в глобальной переменной timerLoadValue для последующего использования в ISR.

Наконец, Setup() выводит timerLoadValue так, чтобы убедиться, что оно в пределах разумного. На этой стадии таймер запущен и процедура ISR вызывается с заданной частотой. Если подключить осциллограф, можно увидеть переключение вывода, генерирующее частоту, равную 1/2 интервала таймера. 1/2 получается потому, что порт устанавливается в низкий уровень в одной ISR и в высокий уровень – в другой.

Главная программа. Функция Loop()

Эта функция вызывается снова и снова, пока программа запущена. Каждый раз при возврате из цикла функция вызывается снова. В листинге 2.4 приведен текст основного цикла программы.

Необходимо отметить, что функция цикла ничего не должна делать относительно переключения линии ввода/вывода. Все это управляется ISR, позволяющей функции цикла заняться другими задачами, не обращая внимания на процессы, происходящие в ISR. Это одно из достоинств программ, управляемых прерываниями. Функции вызываются при появлении события и отделены от вашей прикладной программы.

Листинг 2.4

```
void loop(void) {
// Собирает задержку ISR каждые 10 мс.
delay(10);
// Собирает текущее значение задержки из ISR и увеличивает счетчик на 1
// the sample counter
latencySum+=latency;
sampleCount++;
```

```
// Как только наберется 10 замеров, вычисляет и выводит результат измерений
if(sampleCount>99) {
float latencyAverage;
float loadPercent;
// Вычисляет среднюю задержку
latencyAverage=latencySum/100.0;
// Обнуляет значения сумм
sampleCount=0;
latencySum=0;
// Вычисляет ожидаемый процент загрузки процессора
loadPercent=latencyAverage/(float)timerLoadValue;
loadPercent*=100; //Переводит доли в проценты;
// Выводит среднюю задержку
Serial.print("Latency Average:");
Serial.print((int)latencyAverage);
Serial.print(".");
latencyAverage-=(int)latencyAverage;
Serial.print((int)(latencyAverage*100));
// Выводит ожидаемый процент загрузки
Serial.print("Load:");
Serial.print((int)loadPercent);
Serial.println("%");
}
}
```

Функция цикла начинается с задержки на 10 мс. Задержка на 10 мс регулирует то, как часто производятся замеры задержки и вывод измерений. После того как было произведено 100 измерений, каждое из которых занимает 10 мс, результат выводится на экран.

Далее значение задержки из ISR накапливается в глобальной переменной latencySum. Кроме того, счетчик накопленного количества замеров увеличивается на единицу.

Далее проверяется, было ли уже накоплено 100 замеров. Если накопилось 100 замеров, то вычисляется среднее значение делением накопленной задержки на число замеров и результат сохраняется в latencyAverage. После этого очищаем аккумулятор и счетчик замеров, чтобы можно было начать все заново.

После измерения задержки можно подсчитать ожидаемую загрузку процессора. Это значение подсчитывается и выводится как результат измерения, так что можно оценить влияние программы ISR.

В примере программы таймер загружен шестнадцатеричным значением D4 или десятичным 212. Это означает, что он будет прерван каждый раз, как только отсчитает 44 такта. Пока процессор выполняет код ISR, таймер отсчитывает около 20 тактов, так что остается всего около 24 тактов до того, как он снова вернется в ISR. Эти 24 такта – все время, которое основная программа получает на выполнение. Так что из общего времени 44 такта между прерываниями 20 тактов потрачено на ISR, оставляя около 24 тактов прикладной программе. Это составляет около 45 % времени, потраченного процессором на ISR [10].

Глава 3. ПРОГРАММИРОВАНИЕ БЕСПРОВОДНЫХ СИСТЕМ СВЯЗИ

3.1. Основные характеристики модулей беспроводной передачи данных

Трансивер – электронное устройство, которое представляет собой комбинацию радиопередатчика и радиоприемника в одном устройстве или микросхеме. Радиотрансиверы широко используются в беспроводных системах связи и организуют двухстороннюю радиосвязь.

Применение радиочастотных модулей в основном связано с малообъемными и среднеобъемными продуктами для потребительских приложений, таких как беспроводные системы сигнализации, открыватели гаражных ворот, интеллектуальные сенсорные приложения, беспроводные системы домашней автоматизации, промышленные пульты дистанционного управления и т. п.

Существуют радиомодули, организующие полудуплексный и полнодуплексный режимы работы. При организации полудуплексного режима работы устройство в один момент времени может либо только передавать, либо только принимать информацию. В таком случае при осуществлении связи в радиомодуле происходит коммутация между приемником и передатчиком в зависимости от направления передачи. Это позволяет передатчику и приемнику быть соединенными с одной и той же антенной и обеспечивает защиту передатчика от повреждения приемника. С помощью такого типа приемопередатчика для осуществления ответной передачи необходимо сначала дождаться окончания приема сообщения.

При организации полнодуплексного способа связи устройство может в любой момент времени передавать и принимать информацию. Передача и прием ведутся устройством одновременно по двум физически разделенным каналам связи на двух различных частотах.

Беспроводные радиочастотные трансиверы применяются для телефонной связи, радиостанций, в рациях, смартфонах, мобильниках, для беспроводных устройств. Основной особенностью радиочастотного модуля является встроенная система для беспроводной связи с другим устройством.

В настоящее время на рынке присутствует широкий ассортимент трансиверов в различном исполнении. Трансиверы применяются для приема-передачи данных между станциями – компьютерами, ЭВМ, серверами, устройствами связи – для транспортировки в сетевой среде или соединения ее элементов. Широкое распространение трансиверы получили в сфере телекоммуникации и технологий. Существующие приемопередатчики можно классифицировать как проводные, оптоволоконные и беспроводные модули передачи данных. Исполнение трансивера может быть в виде отдельного устройства или модуля для разработки телекоммуникационных устройств связи.

Беспроводные трансиверы общего назначения – это микросхемы радиочастотной связи ближнего радиуса действия, работающие в УВЧ-диапазоне частот и обеспечивающие передачу данных на короткие и средние расстояния. Они широко применяются в различных бытовых и технических устройствах, позволяя организовывать беспроводные системы безопасности, мониторинга и домашней автоматизации. Работа ИС первой группы основана на частотной манипуляции (FSK, Frequency Shift Keying), гауссовской FSK (GFSK, Gaussian Frequency Shift Keying), манипуляции с минимальным частотным сдвигом (MSK, Minimum Shift Keying), гауссовской MSK (GMSK, Gaussian Minimum Shift Keying) и амплитудной модуляции включением/выключением (OOK, On/Off Keying).

Беспроводные модули передачи данных представлены на рынке в широком ассортименте и базируются на микросхемах таких фирм, как Analog Device, Atmel, Coronis Systems, Fujitsu, Gransen AS, Honeywell Infineon Technologies AG, Inova, Maxim, Microchip, NEC Electronics, Radiocrafts, RF Monolithics Inc., STMicroelectronics, SunRay, Telecontrolli, Texas Instruments, Wavecom, Xemics и др. В состав этих микросхем обычно входит приемопередатчик (трансивер), управляемый микроконтроллером. Основными техническими характеристиками таких модулей являются рабочая частота, модуляция, поддерживаемое количество каналов, мощность, интерфейс связи и возможности программирования режимов работы.

Для выбора приемопередатчика при разработке беспроводной системы связи необходимо особенное внимание обратить на ряд их важных характеристик.

1. Рабочая частота. Рабочая частота должна находиться в разрешенном диапазоне радиовещательных частот и быть достаточно высокой, чтобы минимизировать размеры антенн и всего устройства в целом. Оптимальным выбором в данном случае является рабочая частота 2,4 ГГц. Следует отметить, что в этом диапазоне частот могут функционировать и другие системы связи, например Wi-Fi, или разного рода радиоуправляемые устройства. Однако помехи от такого рода систем связи даже будут полезны с учетом специфики лабораторного макета, направленного на повышение достоверности передачи информации.

2. Количество каналов связи. Возможность выбора и перестройки канала связи увеличивает число вариантов применения лабораторного макета. Так, например, при количестве каналов связи больше 32 появляется возможность использовать лабораторный макет для организации радиосвязи с псевдослучайной перестройкой радиочастоты.

3. Вид модуляции. На сегодняшний день в беспроводных модулях передачи данных используется один из следующих видов модуляции: ASK-амплитудная манипуляция сдвигом, FSK-частотная манипуляция сдвигом и OOK модуляция включением – выключением. Необходимо отметить, что вид модуляции не накладывает каких-либо ограничений на применение лабораторного макета, но желательно наличие программируемого выбора вида модуляции.

4. Уровни излучаемой мощности. Наличие возможности программирования уровня излучаемой мощности позволяет расширить функции разрабатываемого лабораторного макета, в том числе и при использовании его в качестве помехоустойчивого кодека.

5. Размер блока передаваемой информации. Для построения помехоустойчивого кодека необходим минимальный блок передаваемой информации не менее 32 байт. Кроме того, возможность программирования заголовка и конца блока расширяет функциональность разрабатываемого устройства.

6. Режим проверки и контроля четности. В существующих трансиверах используется контроль четности, или подсчет контрольной суммы CRC. При выборе приемопередатчика необходима возможность отключения такой проверки для реализации режима помехоустойчивого кодирования.

7. Наличие библиотек программирования трансивера на языке Си++ при использовании для этой цели популярных микроконтроллеров. Наличие готовых библиотек значительно облегчит программирование трансивера и расширит возможности работы с ним.

8. Ценовой диапазон. Цена используемого приемопередатчика не должна быть значительной, что позволит приобретать подобные устройства для обучения студентов.

3.2. Обзор наиболее популярных программируемых модулей беспроводной передачи данных

В качестве примера представленных на рынке приемопередающих модулей, в большой степени удовлетворяющих требованиям, сформулированным в предыдущем подразделе, можно привести ряд трансиверов [13 – 16].

1. Радиомодуль HC-12. Приемо-передающий радиомодуль HC-12 применяется для связи между двумя и более микроконтроллерами, компьютерами на расстоянии до километра. Модуль HC-12 состоит из трансивера SI4463 и контроллера серии STM8S. Устройство представляет собой «беспроводной удлинитель» UART: что было отправлено передатчику – то и передаст контроллеру приемник так, как это происходило бы при проводном соединении. AT-команды, подаваемые на выход SET, позволяют управлять устройством. Основные преимущества такого трансивера: большая дальность действия; тонкая настройка всех параметров; наличие IPEX-разъема. Радиомодуль HC-12 представлен на рис. 3.1.

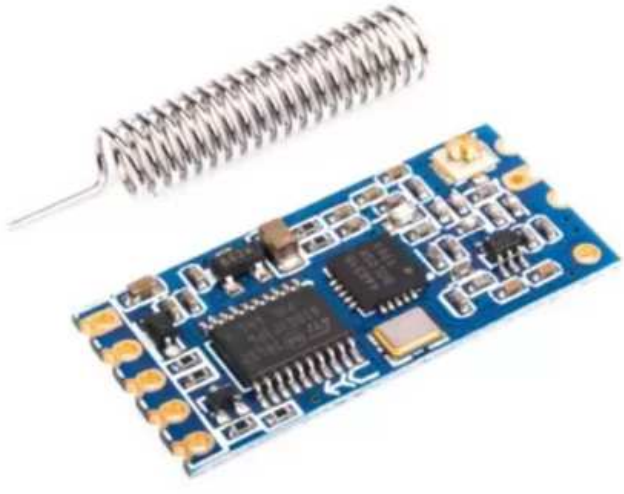

Рис. 3.1. Внешний вид радиомодуля HC-12

2. Радиомодуль NRF24L01. Модули nRF24L01 работают в полудуплексном режиме. NRF24l01 – это высокоинтегрированная микросхема с пониженным потреблением энергии (ULP) 2 Мбит/с для диа-

пазона 2,4 ГГц (рис. 3.2). При помощи модуля можно связать несколько устройств для передачи данных по радиоканалу. Можно объединить до семи приборов в одну общую радиосеть на частоте 2,4 ГГц; один из модулей будет выступать в роли ведущего, остальные – ведомые. Радиомодуль NRF24l01 стоит дешево, поэтому его можно встретить в самых разных проектах – от умного дома до всяких самодельных роботов [5].

Рис. 3.2. Внешний вид радиомодуля nRF24L01

Трансивер NRF24L01 имеет следующие технические характеристики:

- низкие затраты энергии;
- наличие усовершенствованного ускорителя аппаратного протокола ShockBurst;
- операционная система ISM;
- скорость передачи данных 250 Кбит/с, 1 Мбит/с и 2 Мбит/с;
- полная совместимость со всеми стандартными сериями nRF24L Nordic, а также сериями nRF24E и nRF240;
- напряжение питания 3,3 В;
- рабочие температуры от –40 до 85 °С, температуры хранения от –40 до 125 °С;
- дальность связи до 100 м.

NRF24L01 может работать в четырех режимах:

- Power Down;
- Standby;

- RX;
- TX.

Для приема можно задействовать до шести каналов, в один момент времени принимать данные можно только от одного передатчика.

3. Радиомодуль TEA5767FM. Модуль TEA5767FM может работать в режиме поиска радиостанций. Поиск останавливается при нахождении первой станции, имеющей уровень сигнала определенного значения, которое можно изменять. В случае слабого сигнала приемник автоматически переходит в режим моно.

Из дополнительных опций можно назвать корректировку промежуточной частоты, автоматический контроль усиления AGC-цепи, защиту от переполюсовки и др. Малогабаритный микромодуль очень практичен – его можно задействовать во многих современных проектах и начинаниях (например, для реализации сборки цифрового стереорадиоприемника УКВ-FM-диапазона).

Базовые технические параметры такого трансивера:

- напряжение 2,5 – 5 В;
- есть звуковой усилитель для наушников УНЧ TDA1308 (сопротивление 32 Ом);
- ток 70 – 200 мА;
- тип шины I2C;
- рабочие частоты 76 – 108 МГц;
- температурный диапазон –20 ... +80 °C;
- размеры 11 × 12 мм.

Радиомодуль TEA5767FM приведен на рис. 3.3.

Рис. 3.3. Внешний вид радиомодуля TEA5767FM

4. Радиомодуль SI4432. Модули трансивера SI4432 на основе микросхемы SI4432 позволяют реализовать устойчивую двустороннюю радиосвязь в диапазоне частот от 240 до 930 МГц на расстоянии до 1 км на открытой местности и 100 – 300 м – в помещении. Скорость передачи данных 0,123 – 256 Кбит/с, виды модуляции – FSK, GFSK, OOK. Мощность передатчика до +20 dBm. Модули имеют малый размер.

Микросхема имеет такие особенности: таймер автоматического пробуждения; детектор низкого заряда батареи; 64-байтный буфер приема/передачи; автоматический обработчик пакетов; датчик температуры; 8-разрядный АЦП; источник опорного напряжения; три вывода GPIO; различные виды модуляции сигнала (OOK, FSK, GFSK) и др. Микросхема работает от 1,8 до 3,6 В питающего напряжения. Радиомодуль SI4432 приведен на рис. 3.4.

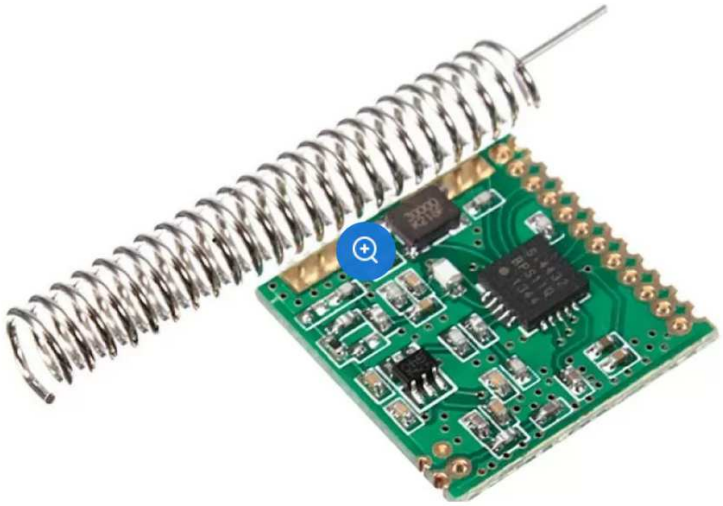

Рис. 3.4. Внешний вид радиомодуля SI4432

5. Беспроводной передатчик FS1000A и приемник MX-RM-5V.

Комплект радиомодулей, состоящий из передатчика (FS1000A) и приемника (MX-RM-5V), предназначен для передачи данных по радиоканалу на частоте 433 МГц. Указанное производителем расстояние уверенного приема – 50 – 100 м в пределах прямой видимости (в зависимости от условий связи и напряжения питания) можно увеличить подключением антенн к передатчику и приемнику. В качестве простейшей антенны можно использовать кусок провода длиной 17 см. Преимуществом данного вида радиомодулей является их дешевизна и

простота подключения к платам Arduino. К недостаткам следует отнести отсутствие обратной связи, низкую скорость передачи и наличие шумов от большого количества других устройств (радиоуправляемых моделей), работающих на этой частоте [9].

Приемник имеет два электрически соединенных входа DATA (использовать можно любой). Приемник обладает высокой чувствительностью и автоматической регулировкой усиления. Из недостатков можно отметить критичность приемника к пульсациям на шине питания, которые он усиливает и принимает за информационный сигнал.

Технические характеристики такого модуля:

- напряжение питания передатчика 3 – 12 В
- напряжение питания приемника 5 В;
- несущая частота 433 МГц;
- потребляемый ток передатчиков 8 мА;
- потребляемый ток приемником 4,5 мА;
- чувствительность приемника –106…–110 dBm;
- выходная мощность передатчика 32 мВт;
- максимальная пропускная способность передатчика 8 Кб/с;
- максимальная пропускная способность приемника 5 Кб/с;
- диапазон рабочих температур –20…+80 °С.

Беспроводной передатчик FS1000A и приемник MX-RM-5V приведены на рис. 3.5.

Рис. 3.5. Внешний вид беспроводного передатчика FS1000A и приемника MX-RM-5V

6. Радиомодуль MBee

MBee подойдут для сложных систем мониторинга, охранных сигнализаций, промышленной связи и т. д. Центр модуля – микросхема CC430 компании Texas Instruments. Она поддерживает протоколы семейства 6LoWPAN и SimpliciTI. Высокая выходная мощность обеспечивает устойчивую передачу данных на 15 – 20 км в зоне прямой видимости. Максимальная скорость обмена данными 500 Кбит/с [10].

Технические характеристики:

- модуль Mbee 868 версии 2.0;
- интерфейс UART;
- рабочий диапазон частот 863 – 873 МГц;
- выходная мощность передатчика 27 dBm;
- чувствительность приемника до −116 dBm;
- скорость передачи данных до 500 Кбит/с;
- напряжение питания 3,3 – 5 В;
- потребляемый ток в режиме передачи до 200 мА;
- потребляемый ток в режиме приема до 50 мА;
- габаритые размеры платы 48,2×22×8 мм.

Рис. 3.6. Внешний вид радиомодуля MBee

Внешний вид радиомодуля MBee приведен на рис. 3.6.

7. Радиомодуль nRF52832. В настоящее время nRF52832 является широко используемым решением, которое представляет собой гибкую и эффективную многопротокольную SoC Bluetooth 5 и Bluetooth mesh. Радиомодуль nRF52832 – это мультипротокольный чип, поддерживающий собственные стеки Bluetooth 5, Bluetooth Mesh, ANT+ и 2,4 ГГц. Он построен на базе процессора ARM Cortex-M4, работающего на частоте 64 МГц и имеющего объемы памяти ROM 512 Кб и RAM 64 Кб [11].

Радиомодуль nRF52832 поддерживает функции Bluetooth 5 и высокую скорость (2 Мбит/с). В кристалле реализованы следующие интерфейсы: I2C, SPI, UART, I2S, PDB, PWM, NFC.

Беспроводной модуль nRF52832 приведен на рис. 3.7.

8. Беспроводной модуль передачи данных VNT9271 имеет следующие характеристики:

- чип контроллера: Атеросы AR9271;
- совместимость со стандартами IEEE 802.11b/g/n;
- работа в диапазоне частот 2,4 ~ 2,5 ГГц в соответствии с мировыми стандартами;
- поддержка 802.11e WMM (Wi-Fi мультимедийный стандарт QoS). Максимальная надежность, пропускная способность и возможность подключения благодаря автоматическому переключению скорости передачи данных;
- поддержка инфраструктурных сетей через точку доступа и специальную сеть посредством пиринговой коммуникации;
- поддержка WEP, WPA, WPA2 и режима безопасности 802.1X;
- поддержка драйверов для Android, Linux, Windows® 8.1, 8, 7, 7 и Windows®;
- техническое описание EMIO-1533;
- совместимые модели ARM: VAB-1000, VAB-820, VAB-800, VAB-600, ARTiGO A600 x86: AMOS-3005, AMOS-3003, ARTiGO A1300, ARTiGO A1300, ARTiGO A1250, VIPRO VP7910, EPIA-E900, EPIA-P910, EPIA-P910, EPIA-P900, EPIA-P900, EPIA-M920, EPIA-M910, EPIA-M900, EPIA-M860, VB700900.

Беспроводной модуль передачи данных VNT9271 приведен на рис. 3.8.

Рис. 3.7. Внешний вид радиомодуля nRF52832

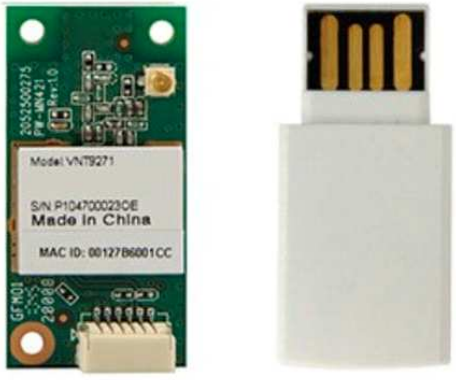

Рис. 3.8. Внешний вид радиомодуля nRF52832

Сравнительные характеристики приведенных модулей в свете сформулированных к ним требований приведены в табл. 3.1.

Таблица 3.1

Сравнительная характеристика приведенных модулей

Тип радиомодуля	Рабочая частота	Кол-во каналов	Режимы мощности, dBm	Модуляция	Блок, байт	CRC	Библиотеки	Цена, руб.
Радиомодуль HC-12	433 МГц	100	От −1 до 20	GFSK	32	Есть	VirtualWire	590
Радиомодуль NRF24L-01	2,400 – 2,527 ГГц	128 (с шагом 1 МГц)	−18, −12, −6, 0	GFSK	32	Есть	RF24	165
Радиомодуль TEA576-7FM	76 – 108 МГц	100	От −1 до 20	AM	32	Нет	Arduino-библиотека TEA5767	208
Радиомодуль SI4432	433,92 МГц	20	−20, −6	FSK, GFSK, OOK	64	Нет	RCSwitch	650
Беспроводной передатчик FS1000A и приемник MX-RM-5V	433 МГц	100	20	AM	32	Нет	VirtualWire, RemoteSwitch, RCSwitch	380
Радиомодуль MBee	863 – 873 МГц	128	−12, +3	OFDM	128	Нет	MBEE_SERIAL Serial1	5990
Беспроводной модуль nRF52832	2,4 ГГц	20	От −20 до +4, с шагом в 4 dBm	AM	64	Есть	VirtualWire	899
Модуль передачи данных VNT9271	2,4 ~ 2,5 ГГц	13	20	OFDM	32	Есть	VirtualWire	3132

Анализируя сравнительные данные из табл. 3.1, следует сделать вывод, что для учебного процесса в качестве учебного пособия для лабораторных и практических работ наиболее целесообразным является выбор беспроводного модуля передачи данных NRF24L01 как самого дешевого и обладающего широкими возможностями по программированию режимов работы со следующими техническими характеристиками:

- встроенная антенна в виде четвертьволнового полоска на плате;
- работа в диапазоне ISM 2,4 ГГц;
- количество радиочастотных каналов – 126;
- общие контакты RX и TX;
- модуляция GFSK;
- скорость передачи данных 1 и 2 Мбит/с;
- неперекрывающееся расстояние между каналами со скоростью 1 Мбит/с – 1 МГц, неперекрывающееся расстояние между каналами со скоростью 2 Мбит/с – 2 МГц;
- программируемая выходная мощность 0, –6, –12 или –18 dBm;
- встроенные фильтры каналов;
- чувствительность –82 dBm при 2 Мбит/с, –85 dBm при 1 Мбит/с;
- программируемое усиление LNA;
- диапазон питания от 1,9 до 3,6 В;
- 4-контактный аппаратный SPI;
- 3 отдельных 32-байтных TX и RX FIFOs.

3.3. Устройство и регистры программирования беспроводного модуля NRF24L01

Модуль состоит из основной микросхемы NRF24L01 и ее обвязки для обеспечения питания и согласования антенны согласно официальной документации. К микроконтроллеру подключаются выводы CE, CSN, SCK MOSI, MISO и IRQ (рис. 3.9).

Вся конфигурация радиомодуля определяется значениями в некоторых регистрах конфигурации. Все эти регистры доступны для записи через интерфейс SPI.

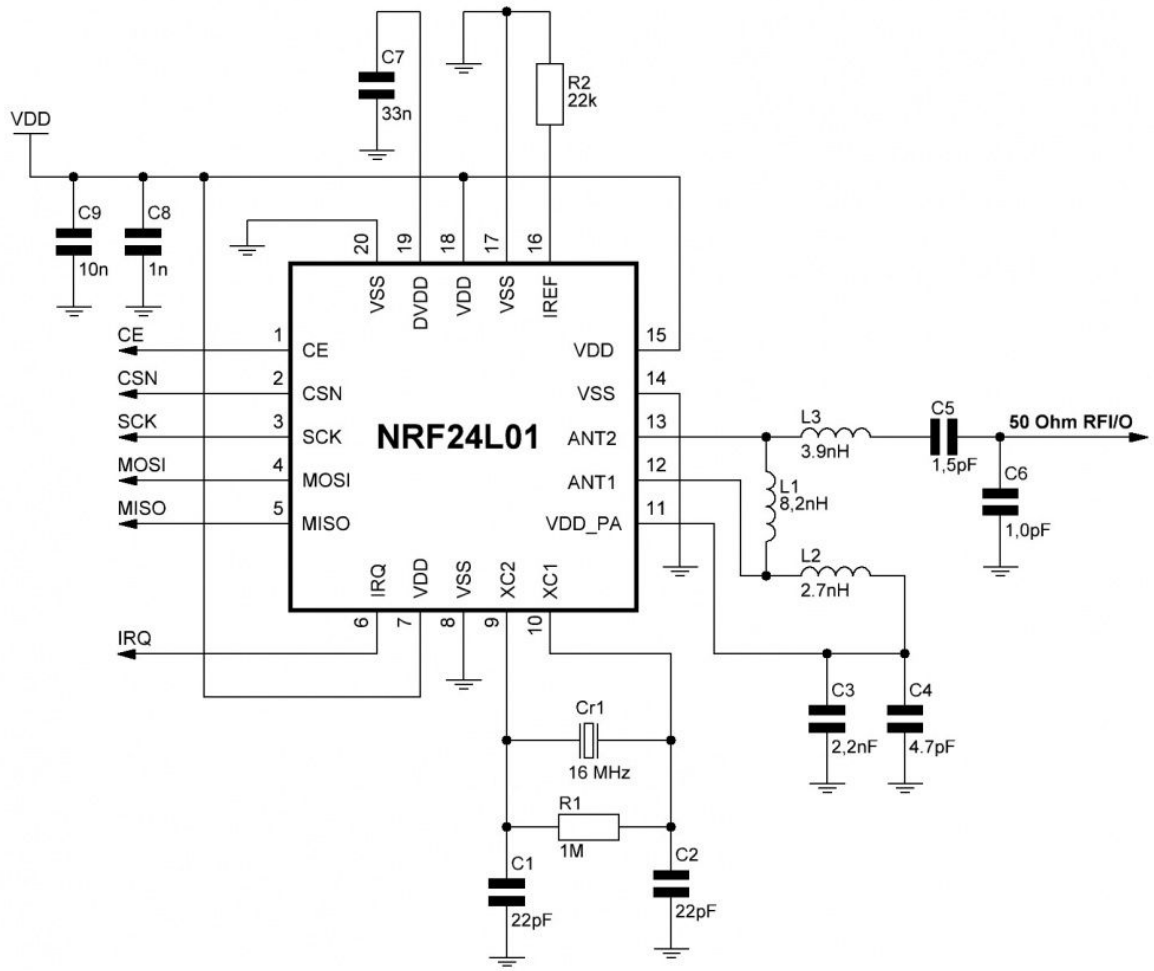

Рис. 3.9. Электрическая схема радиомодуля на микросхеме NRF24L01

SPI (Serial Peripheral Interface) – стандарт полнодуплексной последовательной синхронной передачи данных, разработанный для обеспечения простого, недорогого и высокоскоростного интерфейса микроконтроллеров и периферийных устройств. SPI также иногда называют четырехпроводным интерфейсом. В SPI используются четыре цифровых сигнала:

MOSI – выход ведущего, вход ведомого (англ. Master Out Slave In). Служит для передачи данных от ведущего устройства ведомому.

MISO – вход ведущего, выход ведомого (англ. Master In Slave Out). Служит для передачи данных от ведомого устройства ведущему.

SCLK или SCK – последовательный тактовый сигнал (англ. Serial Clock). Служит для передачи тактового сигнала для ведомых устройств.

CSN или SS – выбор микросхемы, выбор ведомого (англ. Chip Select, Slave Select).

Доступные команды для использования в интерфейсе SPI показаны в табл. 3.2 [15, 16]. Всякий раз, когда вывод CSN имеет низкий уровень, модуль ожидает инструкции. Каждая новая инструкция должна начинаться переходом с высокого уровня на низкий на выводе CSN. Команды последовательного переключения SPI имеют формат [Там же]:

– один байт инструкции: от старшего бита к младшему (MSBit to LSBit);

– байты данных: от младшего байта к старшему (LSByte to MSByte), старший бит в каждом байте первый (MSBit to LSBit).

Датаграммы процессов чтения и записи в регистры модуля через интерфейс SPI показаны на рис. 3.10, 3.11.

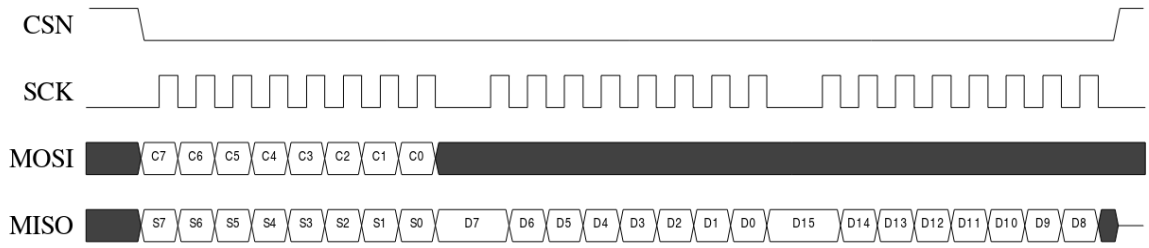

Рис. 3.10. Датаграмма операции чтения через интерфейс SPI

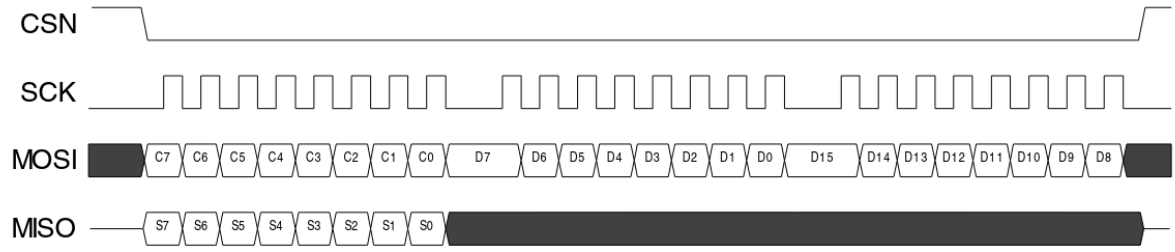

Рис. 3.11. Датаграмма операции записи через интерфейс SPI

Таблица 3.2

Набор инструкций для интерфейса SPI модуля nRF24L01

Наименование инструкции	Двоичный формат инструкции	Количество байт данных	Операция
R_REGISTER	000A AAAA	От 1 до 5, младший байт первый (LSB)	Инструкция чтения регистров. AAAAA – 5 бит адреса памяти регистра
W_REGISTER	001A AAAA	От 1 до 5, младший байт первый (LSB)	Инструкция записи в регистры. AAAAA – 5 бит адреса памяти регистра
R_RX_PAYLOAD	0110 0001	От 1 до 32, младший байт первый (LSB)	Чтение полезной нагрузки принимаемого сообщения: 1 – 32 байта. Операция чтения всегда будет начинаться с байта 0. Полезная нагрузка будет удалена из регистра FIFO после ее чтения
W_TX_PAYLOAD	1010 0000	От 1 до 32, младший байт первый (LSB)	Запись полезной нагрузки отправляемого сообщения: 1 – 32 байта. Операция записи всегда будет начинаться с байта 0
FLUSH_TX	1110 0001	0	Очистка регистра FIFO передачи. Используется в режиме передачи
FLUSH_RX	1110 0010	0	Очистка регистра FIFO приема. Используется в режиме приема. Не должна выполняться во время передачи подтверждения
REUSE_TX_PL	1110 0011	0	Используется для повторной отправки полезной нагрузки. Пакеты будут повторно отправляться до тех пор, пока CE имеет высокий уровень
NOP	1111 1111	0	Может использоваться для чтения регистра STATUS

Для отправки пакета данных до приемного модуля, а также для приема пришедших данных используется форма сообщения, показанная в табл. 3.3.

Таблица 3.3

Формат отправки пакета данных

Преамбула (RX/TX)	Адрес (3 – 5 байт)	Флаги (9 бит)	Полезная нагрузка (1 – 32 байт)	CRC (0 – 2 байт)

Преамбула – используется для обнаружения типа пакета (прием или отправка).

Адрес – поле адреса содержит в себе адрес принимаемого пакета.

Флаги – идентификация пакета 2 бита, которые увеличиваются для каждой новой полезной нагрузки.

Полезная нагрузка – отправляемое или принимаемое сообщение длиной от 1 до 32 байт.

CRC – байты контрольной суммы.

Регистры управления

0x00 CONFIG – регистр конфигурации.

7	6	5	4	3	2	1	0
Reserved	MASK_ RX_DR	MASK_ TX_DS	MASK_ MAX_RT	EN_CRC	CRCO	PWR_UP	PRIM_ RX
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

MASK_RX_DR – бит установки маски прерывания, вызванного регистром приема данных. Если бит установлен в 1 – прерывание не отражается на выводе IRQ, если бит установлен в 0 – прерывание отражается как активное (низкий уровень на выводе IRQ).

MASK_TX_DS – бит установки маски прерывания, вызванного регистром отправки данных. Если бит установлен в 1 – прерывание не отражается на выводе IRQ, если бит установлен в 0 – прерывание отражается как активное (низкий уровень на выводе IRQ).

MASK_MAX_RT – бит установки маски прерывания, вызванного переполнением регистра повторных передач. Если бит установлен в 1 – прерывание не отражается на выводе IRQ, если бит установлен в 0 – прерывание отражается как активное (низкий уровень на выводе IRQ).

EN_CRC – бит включения подсчета контрольной суммы (Cyclic redundancy check, CRC). Если включено автоподтверждение приема путем установки хотя бы одного бита в регистре EN_AA, то значение этого бита устанавливается в единицу автоматически.

CRCO – бит установки размера поля CRC. Если бит установлен в 0 – размер CRC 1 байт, если бит установлен в 1 – размер CRC 2 байта.

PWR_UP – бит включения и выключения питания. Если бит установлен в 0 – питание выключено, если в 1 – питание включено.

PRIM_RX – бит установки режима работы микросхемы. Если бит установлен в 0 – активирован режим передатчика, если бит установлен в 1 – активирован режим приемника.

0x01 EN_AA – регистр установки автоматического ответа.

7	6	5	4	3	2	1	0
Reserved		ENAA_P5 – ENAA_P0					
R/W		R/W					

ENAA_P5 – ENAA_P0 – биты установки автоматического ответа о приеме пакета по соответствующему каналу приема.

0x02 EN_RXADDR – регистр установки каналов приема.

7	6	5	4	3	2	1	0
Reserved		ERX_P5 – ERX_P0					
R/W		R/W					

ERX_P5 – ERX_P0 – биты включения и выключения соответствующих каналов приема.

0x03 SETUP_AW – регистр установки длины поля адреса.

7	6	5	4	3	2	1	0
Reserved						AW	
R/W						R/W	

AW – биты установки длины поля адреса. «00» – запрещенное состояние, «01» – длина адреса 3 байта, «10» – длина адреса 4 байта, «11» – длина адреса 5 байт.

0x04 SETUP_RETR – регистр установки автоматической повторной отправки.

7	6	5	4	3	2	1	0
ARD				ARC			
R/W				R/W			

ARD – биты установки задержки автоматической повторной отправки. «0000» – задержка 250 мкс, «0001» – задержка 500 мкс, «0010» – задержка 750 мкс, … «1111» – задержка 4000 мкс (шаг изменения – 250 мкс).

ARC – биты установки количества автоматических повторных отправок. «0000» – автоматическая повторная отправка отключена, «0001» – однократный повтор отправки … «1111» – до 15 повторов отправки.

0x05 RF_CH – регистр установки частоты канала.

7	6	5	4	3	2	1	0
Reserved	RF_CH						
R/W	R/W						

RF_CH – биты установки номера радиоканала от частоты несущей с шагом 1 МГц. Радиочастота несущей вычисляется по формуле 2400 + RF_CH МГц. Допустимые значения – от 0 до 125. При обмене на скорости 2 Мбит/с частота должна отличаться от частоты, используемой другими устройствами, минимум на 2 МГц.

0x06 RF_SETUP – регистр установки радиоканала.

7	6	5	4	3	2	1	0
Reserved			PLL_LOCK	RF_DR	RF_PWR		LNA_HCURR
R/W			R/W	R/W	R/W		R/W

PLL_LOCK – бит установки принудительного сигнала блокировки PLL. Используется только в тесте.

RF_DR – бит установки скорости передачи данных. «0» – скорость 1 Мбит/с, «1» – скорость 2 Мбит/с.

RF_PWR – биты установки выходной мощности в режиме передатчика. «00» – −18 dBm, «01» – −12 dBm, «10» – −6 dBm, «11» – 0 dBm.

LNA_HCURR – бит включения усиления малошумящего усилителя (МШУ).

0x07 STATUS – регистр статуса радиомодуля.

7	6	5	4	3	2	1	0
Reserved	RX_DR	TX_DS	MAX_RT	RX_P_NO			TX_FULL
R/W	R/W	R/W	R/W	R			R

RX_DR – бит прерывания по приему новых данных, устанавливается в «1», когда новые данные поступают в буфер FIFO. Запись «1» очищает бит.

TX_DS – бит прерывания по окончанию отправки данных. Устанавливается в «1», когда пакет данных отправлен. В случае, если установлен регистр автоматического ответа, бит устанавливается в состояние «1» после получения ответа. Запись «1» очищает бит.

MAX_RT – бит прерывания максимального количества повторных попыток передачи. Запись «1» очищает бит. Если установлено значение MAX_RT, его необходимо сбросить, чтобы разрешить дальнейшую связь.

RX_P_NO – биты номера канала данных полезной нагрузки, доступной для чтения из буфера FIFO приема.

TX_FULL – бит флага заполнения буфера FIFO отправки. «1» – буфер заполнен, «0» – есть доступное место в буфере.

0x08 OBSERVE_TX – регистр наблюдения за передачей данных.

7	6	5	4	3	2	1	0
PLOS_CNT				ARC_CNT			
R				R			

PLOS_CNT – биты подсчета потерянных пакетов. Счетчик защищен от переполнения до 15 и прекращает работу на максимальном уровне до сброса. Счетчик сбрасывается записью в RF_CH.

ARC_CNT – биты подсчета повторно отправленных пакетов. Счетчик сбрасывается, когда начинается передача нового пакета.

0x09 CD – регистр обнаружения несущей.

7	6	5	4	3	2	1	0
Reserved							CD
R							R

CD – бит обнаружения несущей устанавливается на высокий уровень, когда внутриполосный РЧ-сигнал обнаружен в режиме приема, в противном случае бит CD имеет низкий уровень.

0x0A RX_ADDR_P0 – регистр данных адреса приема канала 0.

40-битный (5 байт) регистр, используемый для указания адреса канала 0 приемника. Этот канал используется для приема автоподтверждений в режиме передатчика. Автоподтверждения высылаются принимающей стороной с указанием собственного адреса. Поэтому значение этого регистра должно соответствовать значению регистра TX_ADDR для корректной работы в режиме передатчика.

0x0B RX_ADDR_P1 – регистр данных адреса приема канала 1.

40-битный (5 байт) регистр, используемый для указания адреса канала первого приемника. Старшие 4 байта этого регистра являются общими для адресов на каналах 1 – 5.

0x0C – 0x0F RX_ADDR_P2 – RX_ADDR_P5 – регистры данных адреса приема каналов 2 – 5. 8-битные регистры, задающие значения младшего байта адреса для каналов 2 – 5. Значения старших 32 бит берутся из регистра RX_ADDR_P1.

0x10 TX_ADDR – регистр данных адреса передачи.

40-битный (5 байт) регистр, используемый в режиме передатчика в качестве адреса удаленного устройства. При включенном режиме автоподтверждения удаленное устройство ответит подтверждением с указанием своего же адреса. Это подтверждение принимается на ка-

нале 0, поэтому для успешной передачи значение регистра RX_ADDR_P0 должно быть идентично этому.

0x11 – 0x17 RX_PW_P0 – регистры, задающие размер данных, принимаемых по каналам 0 – 5 (регистровая модель аналогична для каналов 1 – 5).

Записываемое число соответствует длине принимаемого пакета (от 1 до 32 байт).

7	6	5	4	3	2	1	0
Reserved		RX_PW_P0					
R/W		R/W					

0x17 FIFO_STATUS – регистр статуса буфера FIFO.

7	6	5	4	3	2	1	0
Reserved	TX_REUSE	TX_FULL	TX_EMPTY	Reserved		RX_FULL	RX_EMPTY
R/W	R	R	R	R/W		R	R

TX_REUSE – бит, показывающий готовность повторной отправки последнего пакета данных, если установлено значение «1». Пакет будет повторно отправлен до тех пор, пока вывод CE имеет высокий уровень. TX_REUSE устанавливается инструкцией SPI REUSE_TX_PL и сбрасывается инструкцией W_TX_PAYLOAD или FLUSH TX.

TX_FULL – бит флага заполнения буфера FIFO передачи. «1» – FIFO заполнен, «0» – в буфере доступно место.

TX_EMPTY – бит, показывающий, что буфер FIFO передачи пуст. «1» – буфер пуст, «0» – в буфере находятся данные.

RX_FULL – бит флага заполнения буфера FIFO приема. «1» – FIFO заполнен, «0» – в буфере доступно место.

RX_EMPTY – бит, показывающий, что буфер FIFO приема пуст. «1» – буфер пуст, «0» – в буфере находятся данные.

3.4. Подключение к микроконтроллеру

Подключение модуля к микроконтроллеру осуществляется согласно распиновке, приведенной на рис. 3.12. Во время инициализации микросхема NRF24L01 потребляет значительный ток, который не может обеспечить стандартный преобразователь Arduino напряжением 3 В. В результате наблюдаются помехи радиосвязи. Избавиться от них поможет электролитический конденсатор емкостью около 100 мкФ, который подключается параллельно контактам GROUND и VCC модуля. Дополнительная емкость помогает сгладить пульсации при запуске и обеспечивает достаточную мощность [15 – 17].

Законченное подключение модуля NRF24L01 к отладочной плате Arduino Nano показано на рис. 3.13.

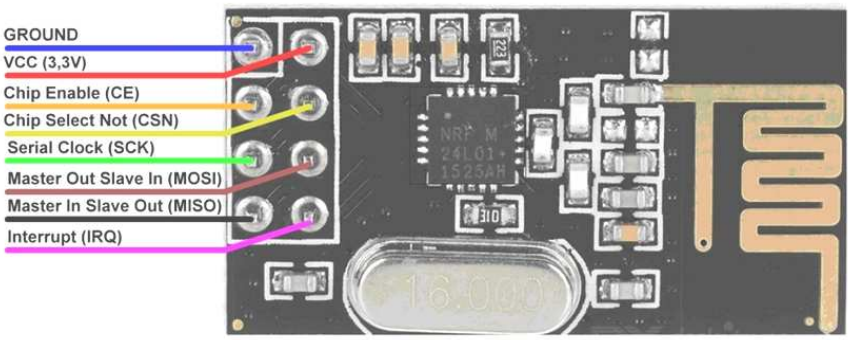

Рис. 3.12. Распиновка модуля NRF24L01 для подключения к микроконтроллеру

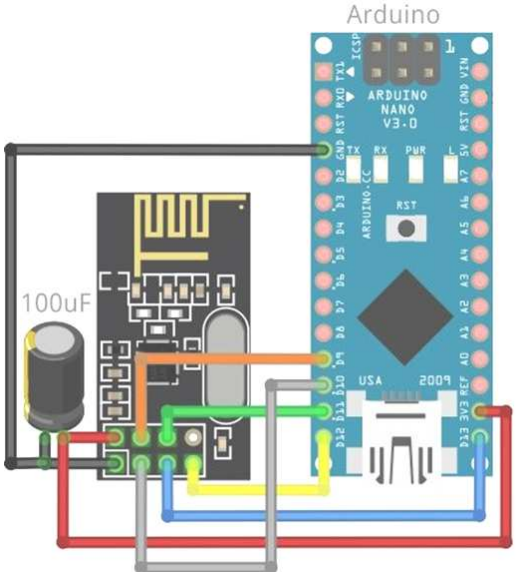

Рис. 3.13. Подключение модуля NRF24L01 к отладочной плате Arduino Nano

3.5. Применение микроконтроллеров для управления программируемыми трансиверами

В составе интегрированной среды разработки ArduinoIDE имеется библиотека RF24 для управления модулем NRF24L01. Библиотека написана на языке программирования C++ и реализована в виде одного класса. При создании экземпляра класса в пользовательском коде в качестве аргументов в конструктор класса передаются номера выводов, к которым подключены выводы CE и CSN радиомодуля. На листинге 3.1 представлен пример инициализации модуля.

Листинг 3.1. Пример кода инициализации модуля NRF24L01 с использованием библиотеки RF24.h

```
#include <SPI.h>
#include <nRF24L01.h>
#include "RF24.h"

RF24 radio(10, 9);
const uint32_t pipe = 111156789;

void setup()
{
  radio.begin();
  radio.setDataRate(RF24_1MBPS);
  radio.setCRCLength(RF24_CRC_8);
  radio.setChannel(111);
  radio.setAutoAck(false);
  radio.openReadingPipe(1, pipe);
  radio.startListening();
}
```

В строке 5 листинга 3.1 выполняется создание экземпляра класса RF24 с указанием в качестве параметров номеров выводов, к кото-

рым подключены выводы CE и CSN радиомодуля. Далее в функции инициализации «void setup()» методом begin() запускается инициализация модуля.

Методом «setDataRate» в строке 9 листинга устанавливается скорость передачи данных от передатчика приемнику. В качестве параметров передаются определения RF24_1MBPS или RF24_2MBPS, что соответствует скоростям 1 Мбит/с и 2 Мбит/с.

Функцией «setCRCLength» устанавливается длина поля контрольной суммы. В качестве параметров можно передать определения «RF24_CRC_8» – длина поля 8 бит, «RF24_CRC_16» – длина поля 16 бит или «RF24_CRC_DISABLED» – проверка контрольной суммы отключена.

Установка частоты канала для обмена данными осуществляется через метод «setChannel». В качестве аргументов функции передается число от 0 до 125, что соответствует количеству смещения частоты канала от несущей на 1 МГц. Например, несущая частота модуля равна 2400 МГц, в качестве аргумента функции передано число «111» (строка 12 листинга 3.1), соответственно частота канала будет равна 2400 МГц + 111 · 1 МГц = 2511 МГц.

Методом «setAutoAck» включается или выключается прием автоподтверждения приема пакета данных.

Для открытия прослушивания определенного радиоканала модуля вызывается метод «openReadingPipe» с указанием номера канала и его адреса в виде 40-битного числа.

Начало прослушивания выбранного канала запускается методом «startListening».

На данном этапе инициализация модуля завершена. Далее необходимо ожидать приема данных от передатчика и заниматься их обработкой. В листинге 3.2 приведен пример чтения данных в основном цикле программы.

В строке 4 листинга 3.2 методом «available()» проверяется наличие доступных для чтения данных, и в случае, если они есть, методом «read» выполняется их чтение в байтовый массив «data» размером 32 байта.

Листинг 3.2. Пример чтения данных в основном цикле программы из радиомодуля NRF24L01

```
byte data[32];

void loop()
{
  if(radio.available())
        {
          radio.read(data, 32);
          Serial.print("Принято сообщение: ");
          Serial.println((char*)data);
         }
}
```

Далее для наглядности принятых данных осуществляется их вывод в консоль методом «Serial.println».

Следует отметить возможность использования функции radio.writeAckPayload. Эта функция позволяет подтвердить прием данных, т. е. приемник на какой-то момент становится передатчиком и по радиоканалу сообщает об успешном приеме информации передатчику. Тот, в свою очередь, после передачи блока данных становится приемником и какое-то время ждет подтверждения приема. Подобный режим работы (ожидание подтверждения принятой информации) задается при программировании передающего устройства.

Инициализация передающего устройства немного отличается и показана в листинге 3.3. Добавляется настройка выходного уровня передатчика.

Листинг 3.3. Пример кода инициализации модуля NRF24L01 в режиме передатчика с использованием библиотеки RF24.h

```
#include <SPI.h>
#include <nRF24L01.h>
#include "RF24.h"
```

```
RF24 radio(10, 9);
const uint32_t pipe = 111156789;

void setup()
{
 radio.begin();
 radio.setDataRate(RF24_1MBPS);
 radio.setCRCLength(RF24_CRC_8);
 radio.setChannel(111);
 radio.setAutoAck(false);
 radio.setPALevel(RF24_PA_HIGH);
 radio.openWritingPipe(pipe);
}
```

В строке 14 листинга 3.3 вызывается метод установки уровня выходной мощности передатчика «setPALevel». В качестве аргументов в него могут передаваться следующие определения:

RF24_PA_MIN – уровень −18 dBm на высокочастотном выходе модуля;

RF24_PA_LOW – уровень −12 dBm на высокочастотном выходе модуля;

RF24_PA_HIGH – уровень −6 dBm на высокочастотном выходе модуля;

RF24_PA_MAX – уровень −0 dBm на высокочастотном выходе модуля.

Для открытия радиоканала передачи модуля вызывается метод «openWritingPipe» с указанием его адреса в виде 40-битного числа.

После инициализации выполнение программы переходит в основной цикл (функция «loop»), в котором выполняется отправка данных в модуль для передачи по радиоканалу. В листинге 3.4 приведен пример кода программы, которая считывает сообщение из последовательного порта и отправляет в радиомодуль функцией «write».

Листинг 3.4. Пример отправки считанных из последовательного порта данных в радиомодуль NRF24L01

```
byte readData[32];

void loop()
{
  if (Serial.available())
                {
                  int recvSize = Serial.readBytes(readData,32);
                  radio.write(potValue, recvSize);
                 }
}
```

В строке 4 листинга 3.4 выполняется проверка доступных данных последовательного порта функцией «Serial.available». Если в порте имеются данные, происходит считывание их в байтовый массив «readData» и отправка в радиомодуль для последующей передачи по радиоканалу.

Обработку информации модуль NRF24L01 проводит с помощью отправки контрольной суммы для помехоустойчивого приема. Перед отправкой данных передатчик суммирует все байты и отправляет их, например, последними в посылке. Благодаря переполнению переменной даже в один байт можно бесконечно суммировать данные и получить в итоге конкретное число, отражающее все передаваемые данные. Приемник получает посылку, в свою очередь, суммирует все байты кроме последнего (или двух последних, если контрольная сумма 16-битная), а затем сравнивает это значение с полученной контрольной суммой. Если отличается хоть на единицу – данные повреждены. Причем поврежденными при передаче могут быть как сами данные, так и контрольная сумма: в любом случае они не совпадут, а это значит, что передача произошла с ошибкой. Эту функцию можно отключить для того, чтобы добавить собственный код коррекции ошибок и провести анализ ошибочного приема пакетов данных. Для этого есть команда radio.disableCRC() [18 – 29]. В листинге 3.5 приведен пример программы сканирования каналов.

Листинг 3.5. Программа для сканирования каналов:

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
RF24 radio(7,8); // инициализировать модуль на пинах 7 и 8 для нашего микроконтроллера
const uint8_t num_channels = 128;
uint8_t values[num_channels];
void setup(void)
{
  Serial.begin(9600);
  printf_begin();
  radio.begin();
  radio.setAutoAck(false);
  radio.startListening();
  radio.printDetails();    // Функция напечатает в окно порта информацию о трансивере и адресе его памяти, это укажет, что трансивер распознан микроконтроллером
 delay(5000);    // Задержка 5 с перед началом сканирования
 radio.stopListening();
  int i = 0;   // Данная функция напечатает заголовки всех 128 каналов
  while (i < num_channels ) {
  printf("%x",i>>4); ++i;
    }
   printf("\n\r");
   i = 0;
   while ( i < num_channels ) {
      printf("%x",i&0xf);
      ++i;
    }
   printf("\n\r");
 }
const int num_reps = 100;
void loop(void)
```

```
{
  memset(values,0,sizeof(values));
  int rep_counter = num_reps;
  while (rep_counter--) {
   int i = num_channels;
   while (i--) {
     radio.setChannel(i);
     radio.startListening();
    delayMicroseconds(128);
    radio.stopListening();
    if ( radio.testCarrier() )
    ++values[i];
  }
 }
 int i = 0;
 while ( i < num_channels ) {
 printf("%x",min(0xf,values[i]&0xf));
 ++i;
  }
  printf("\n\r");
}
int serial_putc( char c, FILE * ) {
  Serial.write( c );
  return c;
}
void printf_begin(void) {
  fdevopen( &serial_putc, 0 );
}
```

Пример работы программы приведен на рис. 3.14.

После информации о трансивере первые две строчки обозначают названия каналов в столбик. Строчки ниже показывают зашумленность канала в шестнадцатеричном виде. Эти строчки обновляются и записываются одна под другой. В данном случае первые пять строчек показывают зашумленность на 0x00 – 0x20 каналах. Это обусловливается влиянием Wi-Fi-сигнала от роутера частотой 2,4 ГГц.

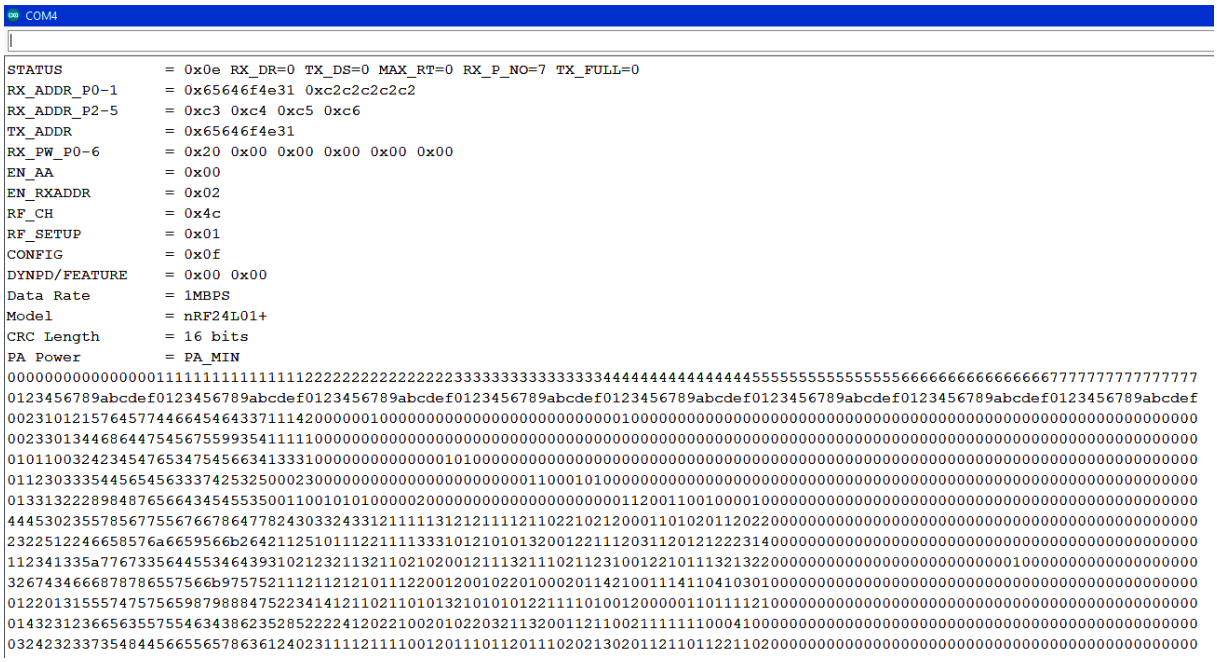

Рис. 3.14. Окно порта микроконтроллера

3.6. Обработка информации в микроконтроллере

Кодек Хемминга. Код Хэмминга – наиболее известный из первых самоконтролирующихся и самокорректирующихся кодов. Коды Хэмминга могут обнаруживать одноразрядные и двухразрядные ошибки или исправлять одноразрядные ошибки без обнаружения неисправленных ошибок [30].

Получение кодового слова выглядит следующим образом:

$$(i_1 \quad i_2 \quad i_3 \quad i_4) \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} = (i_1 \quad i_2 \quad i_3 \quad i_4 \quad r_1 \quad r_2 \quad r_3),$$

где $i_1 - i_4$ – это информация; $r_1 - r_3$ – избыточность.

На вход декодера поступает кодовое слово

$$V = (i'_1,\ i'_2,\ i'_3,\ i'_4,\ r'_1,\ r'_2,\ r'_3),$$

где штрихом помечены символы, которые могут исказиться в результате действия помехи. В декодере в режиме исправления ошибок строится последовательность синдромов:

$$S_1 = r_1 \oplus i_1 \oplus i_2 \oplus i_3,$$
$$S_2 = r_2 \oplus i_2 \oplus i_3 \oplus i_4,$$
$$S_3 = r_3 \oplus i_1 \oplus i_2 \oplus i_4.$$

$S = (S_1,\ S_2,\ S_3)$ – вектор синдромов.

Получение синдрома выглядит следующим образом:

$$(i_1 \quad i_2 \quad i_3 \quad i_4 \quad r_1 \quad r_2 \quad r_3) \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (S_1 \quad S_2 \quad S_3),$$

где S_1, S_2, S_3 – это синдромы. Если они все равны нулю, то ошибок нет. Если они не равны нулю, то двоичное число S_3, S_2, S_1 указывает номер ошибочного бита.

Программы кодера и декодера Хемминга 7-4 на языке Си++ для микроконтроллера Atmega328P на базе отладочной платы Arduino Nano приведены в прил. 1. Работа кодера иллюстрируется на рис. 3.15. Информационное сообщение представляет собой блок данных в виде чисел в диапазоне от 0 до 15. Закодированная информация представлена в нижней строке. Поскольку кодовое слово содержит 7 бит, то закодированные символы представлены числами в диапазоне от 0 до 127.

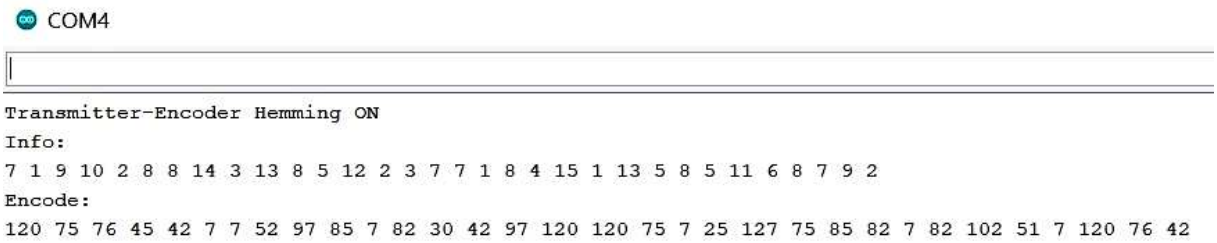

Рис. 3.15. Закодированная информация по коду Хемминга 7-4

На рис. 3.16 представлены результаты обработки декодером закодированной информации и обработки 10 000 информационных блоков, выведены битовая ошибка BER до декодирования и битовая ошибка BER_K после декодирования.

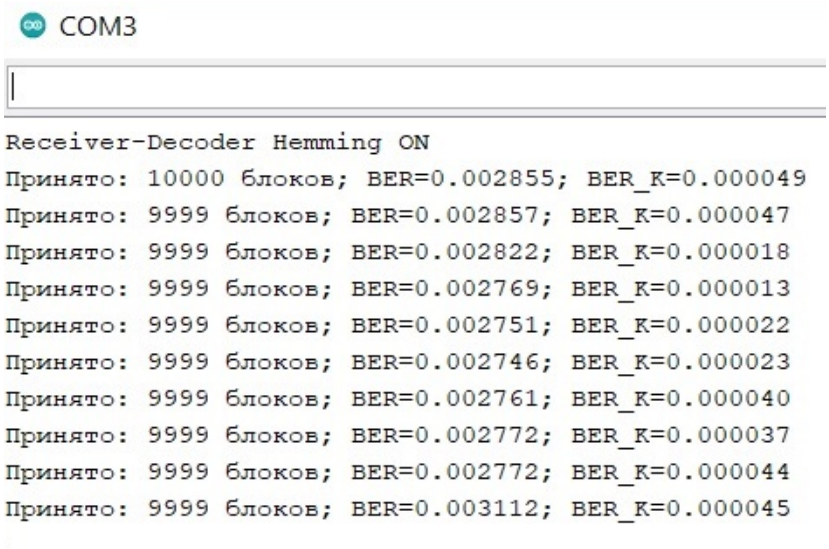

Рис. 3.16. Результаты экспериментальных исследований декодера Хемминга 7-4

Кодек Рида – Соломона. Код Рида – Соломона – это код, который был введен Ирвингом С. Ридом и Гюставом Соломоном. Код Рида – Соломона является подклассом небинарных кодов BCH. Кодер кодов Рида – Соломона отличается от двоичного кодера тем, что он работает с несколькими, а не с отдельными битами. Таким образом, в основном коды Рида – Соломона помогают в восстановлении поврежденных сообщений, которые передаются по сети. В кодах Рида – Соломона мы имеем кодер и декодер [30, 31].

Кодер кодов Рида – Соломона получает данные и перед передачей их по зашумленной сети добавляет некоторые биты четности к исходным битам данных.

С другой стороны, у нас есть декодер кодов Рида – Соломона, который обнаруживает поврежденные сообщения и восстанавливает их из ошибок.

Параметры кода Рида – Соломона:

1. (n, k) код используется для кодирования m-битных символов.
2. Длина блока (n) задается символами $2m - 1$.
3. В кодах Рида – Соломона размер сообщения определяется ($n - 2t$), где t = количество исправленных ошибок.
4. Размер проверки четности задается символами, равными ($n - k$) или $2t$.
5. Минимальное расстояние (a) задается равенством ($2t + 1$).
6. Размер сообщения составляет k бит.

Функция генератора

В кодах Рида – Соломона генераторная функция генерируется с использованием специального полинома, все допустимые кодовые слова в точности делятся на генераторный полином. Функция генератора задается формулой (1).

$$g(x) = (x - a)(x - a2)(x - a3).....(x - a2t). \quad (1)$$

Кодирование

Кодирование в кодах Рида – Соломона выполняется следующими методами:

Код Рида – Соломона с параметрами n (размер блока), k (размер сообщения), q (размер символа в битах). Для кодирования кодируем сообщение как многочлен $p(x)$, а затем умножаем его на многочлен генератора кода $g(x)$, который приведен в формуле (2).

$$g(x) = (x - a)(x - a2)(x - a3)……(x - a2t). \quad (2)$$

Затем сопоставляем вектор сообщения [$x1$, $x2$, … xk] с полиномом $p(x)$ степени $<k$ таким, что $px(ai) = xi$ для всех $i = 1, 2, 3, … k$.

Полином можно сделать с помощью интерполяции Лагранжа.

Отправитель вычисляет $s(x) = p(x)g(x)$, а затем отправляет коэффициенты $s(x)$.

Декодирование

На приемном конце выполняем следующие методы:

1. Приемник получает $r(x)$ на конце приемника.

2. Если $s(x) = = r(x)$, то $r(x)/g(x)$ не имеет остатка.

3. Если он имеет остаток, то $r(x) = p(x)g(x) + e(x)$, где $e(x)$ – полином ошибки.

Декодирование кодов Рида – Соломона представляет собой довольно сложную задачу, решение которой выливается в громоздкий, запутанный и чрезвычайно не наглядный программный код. Типовая схема декодирования (рис. 3.17), получившая название авторегрессионого спектрального метода декодирования, состоит из следующих шагов:

- вычисления синдрома ошибки (синдромный декодер);
- построения полинома ошибки, осуществляемого посредством высокоэффективного, сложно реализуемого алгоритма Берлекэмпа – Мэсси (локатор ошибки);

• нахождения корней данного полинома, обычно решающееся лобовым перебором (корни полинома);

• определения характера ошибки, сводящегося к построению битовой маски, вычисляемой на основе обращения алгоритма Форни или любого другого алгоритма обращения матрицы (характер ошибки);

• исправления ошибочных символов путем наложения битовой маски на информационное слово и последовательного инвертирования всех искаженных битов через операцию XOR (корректор).

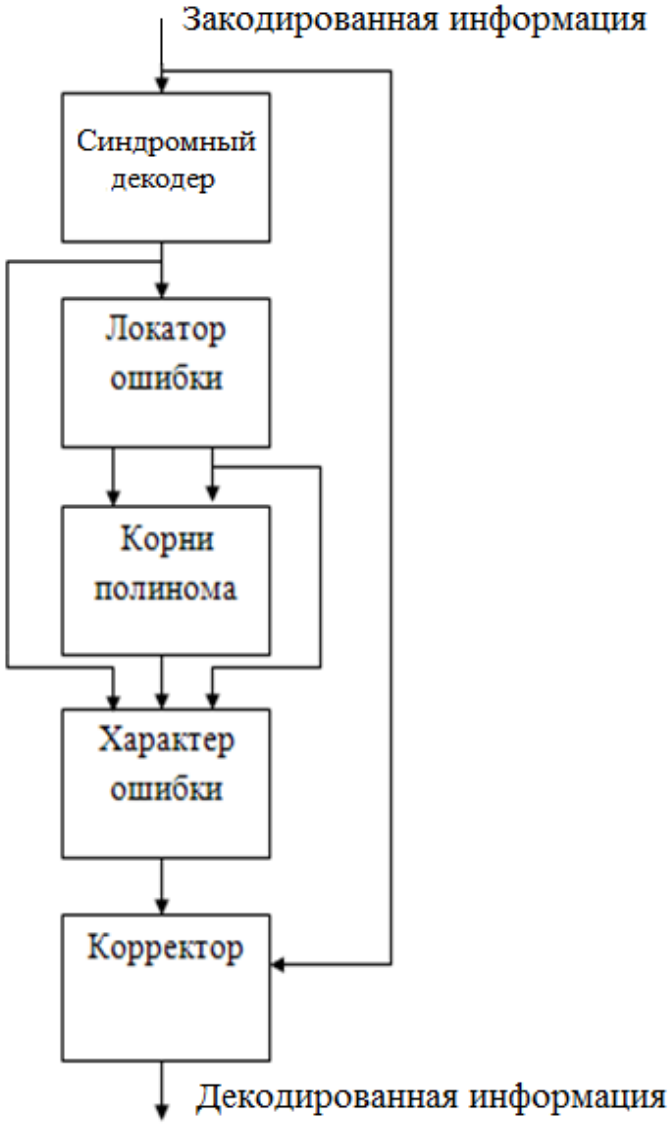

Рис. 3.17. «Жесткий» кодер Рида – Соломона

Полином локатора ошибок вычисляется с помощью алгоритма Берлекэмпа – Месси. Блок-схема алгоритма Берлекэмпа – Месси приведена на рис. 3.18.

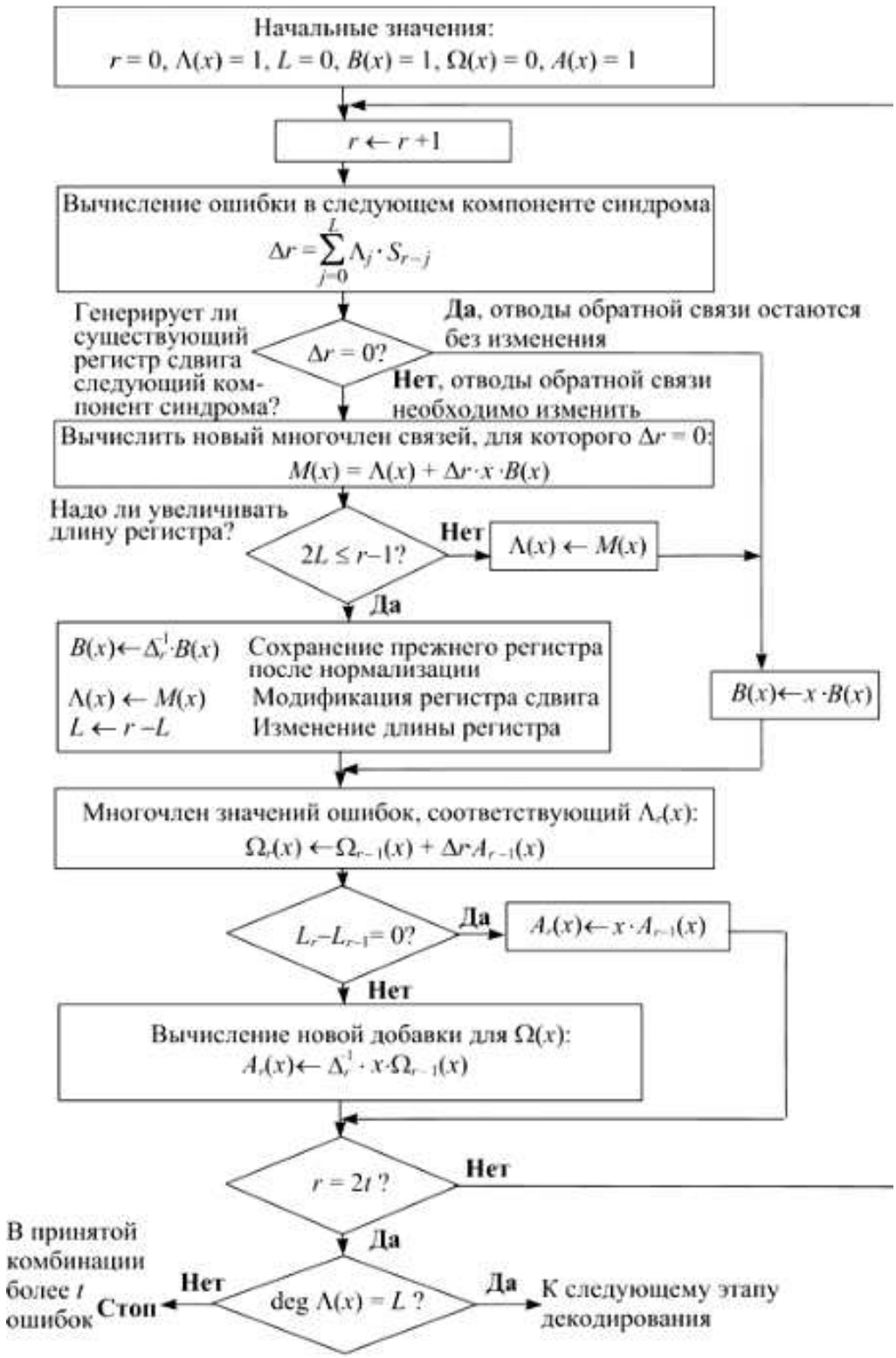

Рис. 3.18. Блок-схема алгоритма Берлекэмпа – Месси: r – номер итерации, $r = 0, 1, 2, \ldots, n - k = 2t$; Sr – компонент синдрома; Δr – ошибка в вычислении S_r (r-я невязка); $L(x)$ – многочлен локаторов ошибок, в соответствии с компонентами которого строится регистр сдвига минимальной длины; L – длина регистра сдвига; $B(x)$ – нормирующая добавка; $W(x)$ – многочлен значений ошибок

Основное содержание алгоритма Берлекэмпа – Месси сформулировано следующим образом: сначала находится самый короткий регистр сдвига, генерирующий S1. Затем проверяют, порождает ли этот регистр также S2. Если порождает, то данный регистр попрежнему остается лучшим решением и нужно проверить, порождает ли он следующие символы синдромного многочлена. На каком-то этапе очередной символ уже не будет генерироваться. В этот момент нужно изменить регистр таким образом, чтобы он правильно предсказывал следующий символ, не менял предсказание предыдущих символов и увеличивал длину регистра на минимально возможную величину [30, 31].

Процесс вычисления продолжается до тех пор, пока не будут порождены первые $2t$ символов синдрома.

Алгоритм строится на основе итеративной процедуры. При каждой итерации должны сохраняться как многочлен связей $L(x)$, так и добавка $B(x)$. Для каждого нового члена $S(x)$ предусматривается проверка правильности предсказания этого символа текущим многочленом связи. Если предсказание правильно, то многочлен связей не меняется, а добавка умножается на x. Если предсказание неправильно, то изменяют текущий многочлен связей, прибавляя к нему добавку. После этого проверяют, увеличилась ли длина регистра. Если она не увеличилась, то текущую добавку оставляют. Если длина регистра возрастает, то лучшей добавкой считают предыдущий многочлен связей. Далее при каждом исправлении эту нормализованную добавку умножают на значение текущей невязки.

Процедура декодирования кодов Рида – Соломона с исправлением ошибок, получившая название быстрого декодирования [4], представлена на рис. 3.19.

Для исправления ошибок декодер выполняет последовательность вычислений, реализующих алгоритм Берлекэмпа – Месси и алгоритм Форни. Алгоритм выполняется за $2t = n - k$ итераций. При этом многочлен значений ошибок находится непосредственным умножением по формуле ключевого уравнения $W(x) = S(x)L(x) \pmod{x2t}$.

Поскольку $W(x)$ находится умножением $L(x)$ на $S(x)$, можно задать последовательность многочленов $Wr(x)$, удовлетворяющих тем же рекуррентным соотношениям, что и многочлены $Lr(x)$. Чтобы из-

бежать путаницы, обозначим добавку при нахождении $W(x)$ через $Ar(x)$. Тогда

$$\Omega_{\gamma+1}(x) = \Omega_{\gamma}(x) + \Delta\gamma A_{\gamma}(x)$$

$$A_{\gamma+1}(x) = \begin{cases} xA_{\gamma}(x), & \text{если } L_{\gamma+1} = L_{\gamma}, \\ x\Omega_{\gamma}(x)\Delta\gamma^{-1}, & \text{если } L_{\gamma+1} > L_{\gamma}. \end{cases}$$

Полагаем, что $W0(x) = 0$ и $A0(x) = 1$.

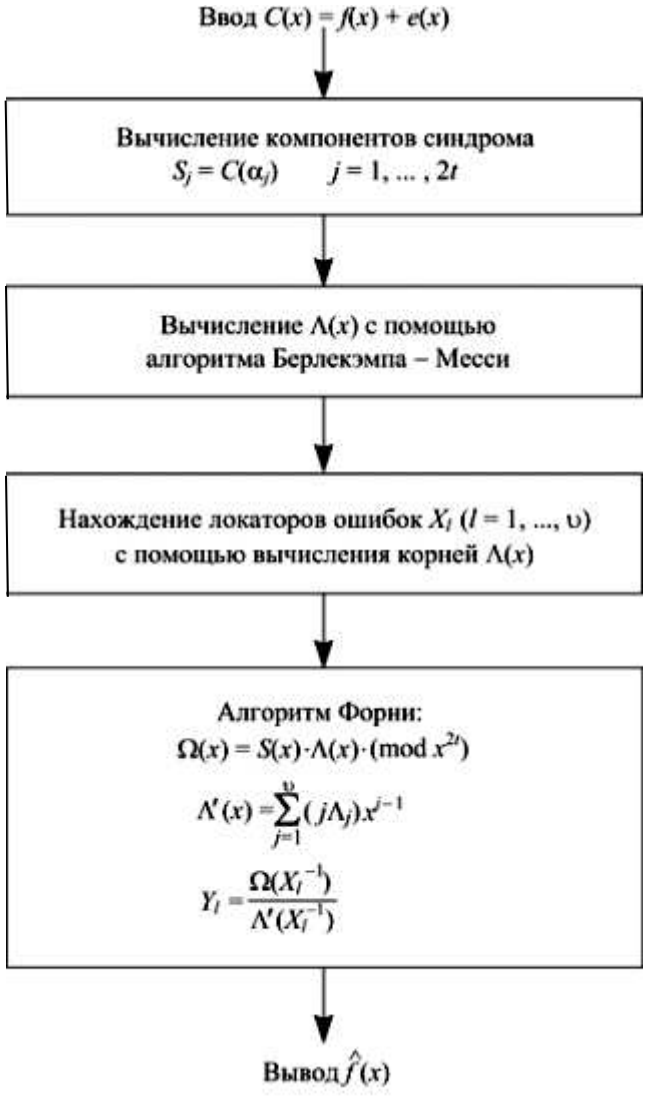

Рис. 3.19. Быстрое декодирование кодов Рида – Соломона

В прил. 2 приведены программы кодера и декодера Рида – Соломона для микроконтроллера на базе платы Arduino и трансивера NRF24L01.

Псевдослучайная перестройка радиочастоты (ППРЧ)

Для обеспечения безопасности и сохранения конфиденциальности при передаче и приеме сообщений используются различные методы защиты информации в системах радиосвязи. Одним из таких методов защиты является псевдослучайная перестройка рабочей частоты (ППРЧ). При данном методе передачи информации по радио несущая частота сигнала меняется в соответствии с псевдослучайной последовательностью чисел, известной как отправителю, так и получателю. Метод повышает помехозащищенность канала связи, который становится устойчивым к глушению сигнала [32 – 35].

Метод ППРЧ используется в спецификации Bluetooth, а схожий метод с более редким изменением частот может применяется в GSM. В последнее время появились идеи симбиоза метода ППРЧ с широкополосными сигналами, например, на основе OFDM для повышения скорости передачи данных [38].

На рис. 3.20 представлена блок-схема алгоритма работы программы передающего модуля.

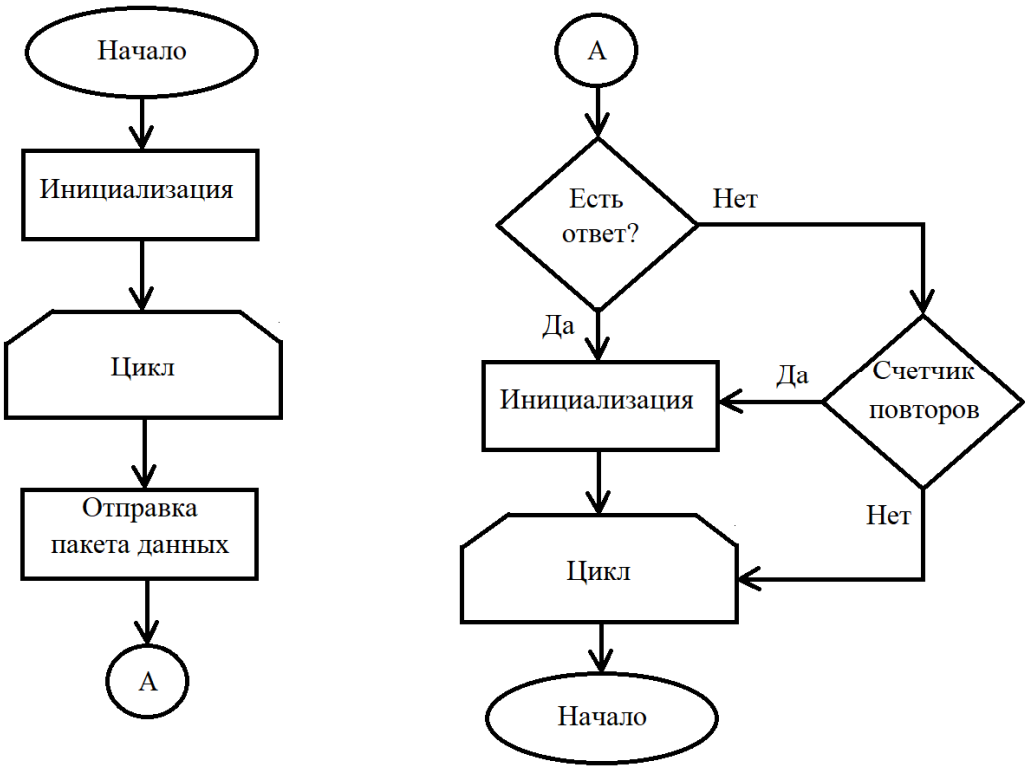

Рис. 3.20. Блок-схема алгоритма работы программы передающего модуля

При подключении питания старт программы начинается с инициализации периферии микроконтроллера, создания псевдослучайной последовательности каналов перестройки рабочей частоты, настройки скорости последовательного порта для вывода информации о переключении каналов и отправления пакета данных, установки первого канала передачи из последовательности.

Далее начинается бесконечный цикл (до отключения питания микроконтроллера) работы основной программы. Программно на радиомодуле включено автоматическое подтверждение о доставке пакета данных до приемника.

Листинг 3.6. Код основного цикла программы передатчика

```
int transmit_cnt = 0;
#define TRANSMIT_TRYING 10

void loop()
{
 if (radio.write(data_to_send, DATA_SIZE))
 {
  printf("Пакет отправлен\n");
  change_channel();
 }
 else
 {
  transmit_cnt++;
  if (transmit_cnt >= TRANSMIT_TRYING)
 {
  change_channel();
  transmit_cnt = 0;
 }
}
delay(50);
}
```

Если при отправке пакета данных передатчик получает информацию о том, что пакет данных получен приемником, то выполняется чтение ответных данных из канала (байт подтверждения). Если байт

подтверждения совпадает с заведомо известным, то передача считается успешной и выполняется переключение радиочастоты на следующую в последовательности. Если ответ не получен и отправить данные не удалось, то передатчик остается на той же частоте и отправляет пакет заново. Если определенное время не удается отправить пакет на текущей частоте, передатчик переключает ее на следующую в последовательности.

Листинг 3.7. Код функции смены каналов

```
int transmit_cnt = 0;
#define TRANSMIT_TRYING 10

void loop()
{
  if (radio.write(data_to_send, DATA_SIZE))
 {
  printf("Пакет отправлен\n");
  change_channel();
 }
 else
 {
   transmit_cnt++;
   if (transmit_cnt >= TRANSMIT_TRYING)
  {
   change_channel();
   transmit_cnt = 0;
  }
 }
 delay(50);
}
```

На рис. 3.21 представлена блок-схема алгоритма работы программы приемного модуля. При подключении питания старт программы начинается с инициализации периферии микроконтроллера, создания псевдослучайной последовательности каналов перестройки рабочей частоты, настройки скорости последовательного порта для

вывода информации о переключении каналов и отправлении пакета данных, установки первого канала приема из последовательности.

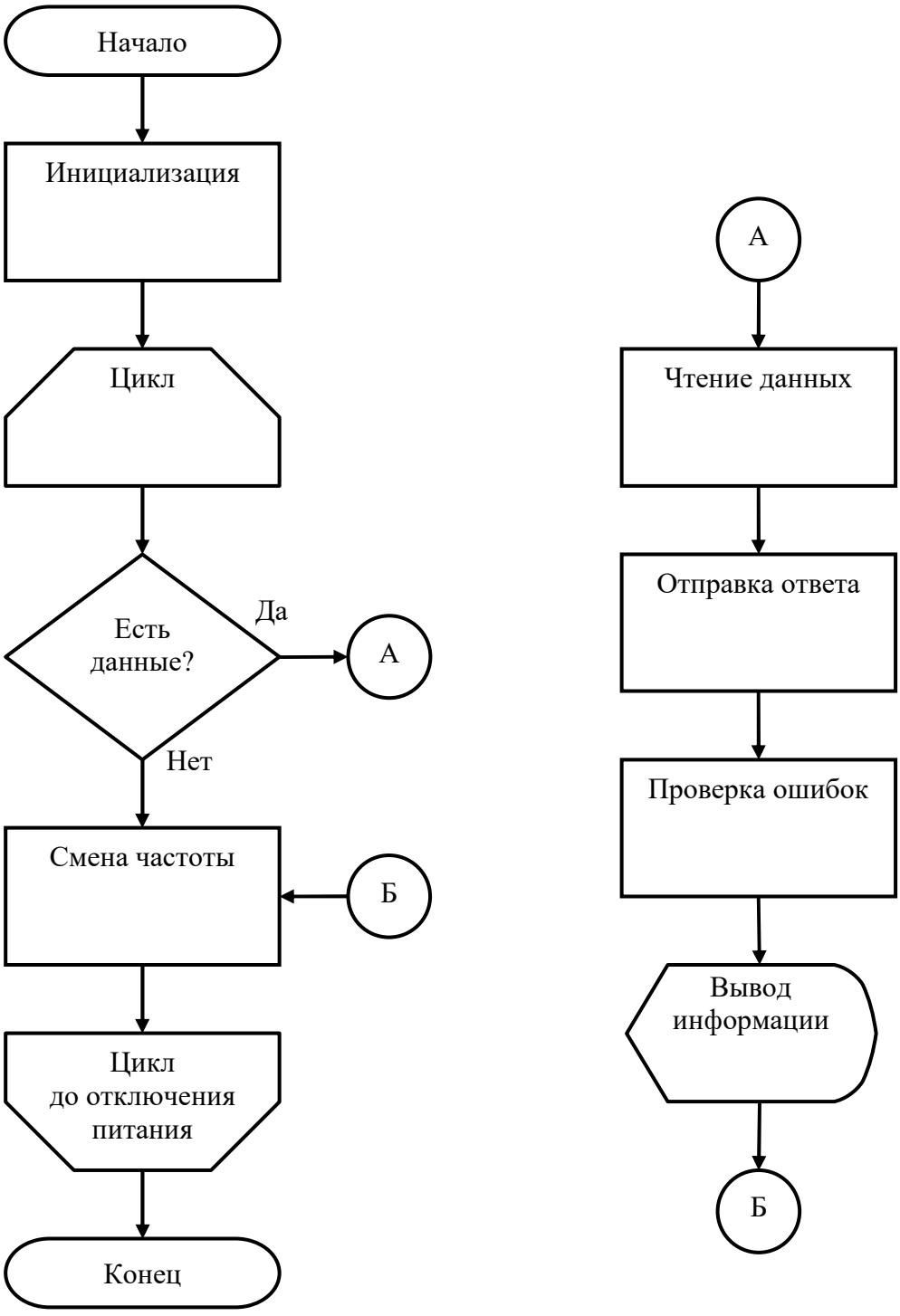

Рис. 3.21. Блок-схема алгоритма работы программы приемного модуля

Основной цикл работы программы заключается в проверке наличия данных в установленном канале. В случае, если данные в канале есть, происходит их считывание в массив, который проходит проверку соответствия отправленным данным. После этого выполняется вывод информации о принятии пакета данных в последовательный порт и количестве ошибок. В независимости от принятия пакета приемник с определенным периодом переключает каналы радиочастоты по заданной последовательности. В случае если синхронизация переключения каналов приемника пропала или на установленном канале появилась узкополосная помеха, то, перебрав все каналы последовательности, приемник установит синхронизацию с передатчиком.

В листинге 3.8 представлен код основного цикла программы приемника.

Листинг 3.8. Код основного цикла программы приемника

```
void loop()
{
 if (radio.available())
 {
  radio.read(data_to_recv, DATA_SIZE);

  uint32_t err_cnt = check_data(data_to_recv);
  printf("Принят массив данных. Количество ошибок: %lu\n", err_cnt);
  memset(data_to_recv, 0, DATA_SIZE);
 }
 change_channel();
 delay(700);
}
```

Приемник и передатчик генерируют псевдослучайную последовательность чисел, которые являются номерами каналов радиомодуля. Между приемником и передатчиком действует обратная связь. Частота передатчика перестраивается в случае успешной отправки или в случае нескольких неудачных попыток отправки. Частота приемника перестраивается через определенный период времени или в случае успешного приема пакета.

При подключении питания после инициализации модулей выводится информация о номерах каналов и начинается процесс синхронизации (рис. 3.22, 3.23).

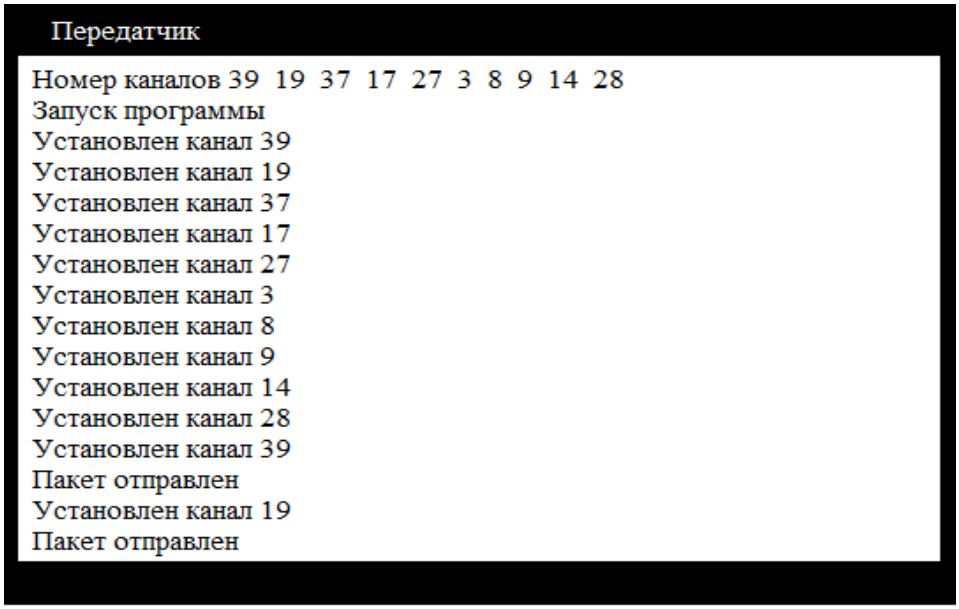

Рис. 3.22. Снимок экрана вывода информации с монитора порта передатчика в момент включения питания устройства

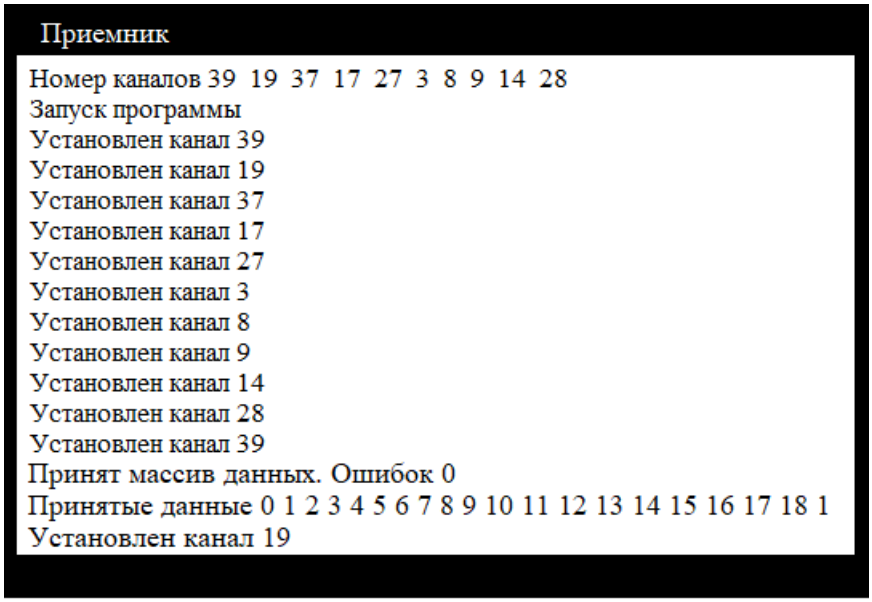

Рис. 3.23. Снимок экрана вывода информации с монитора порта приемника в момент включения питания устройства

На рис. 3.24, 3.25 показан вывод информации в случае успешной передачи и приема модулей в процессе работы, а также периодически возникающие пропуски каналов псевдослучайной последовательности.

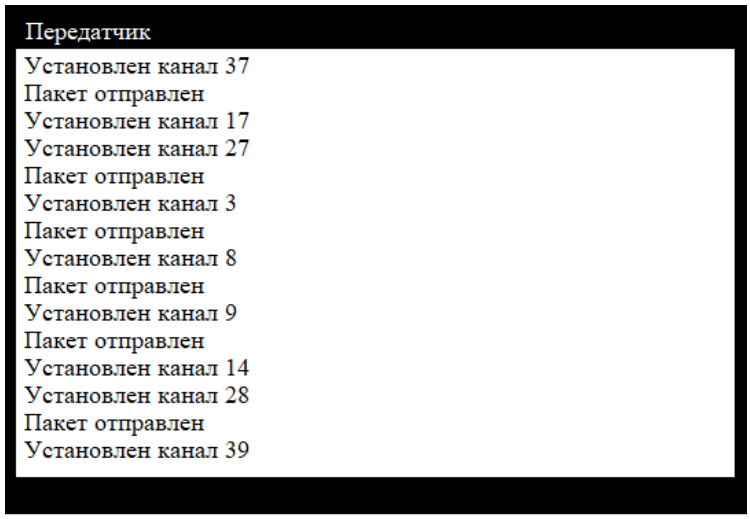

Рис. 3.24. Снимок экрана вывода информации
с монитора порта передатчика во время работы

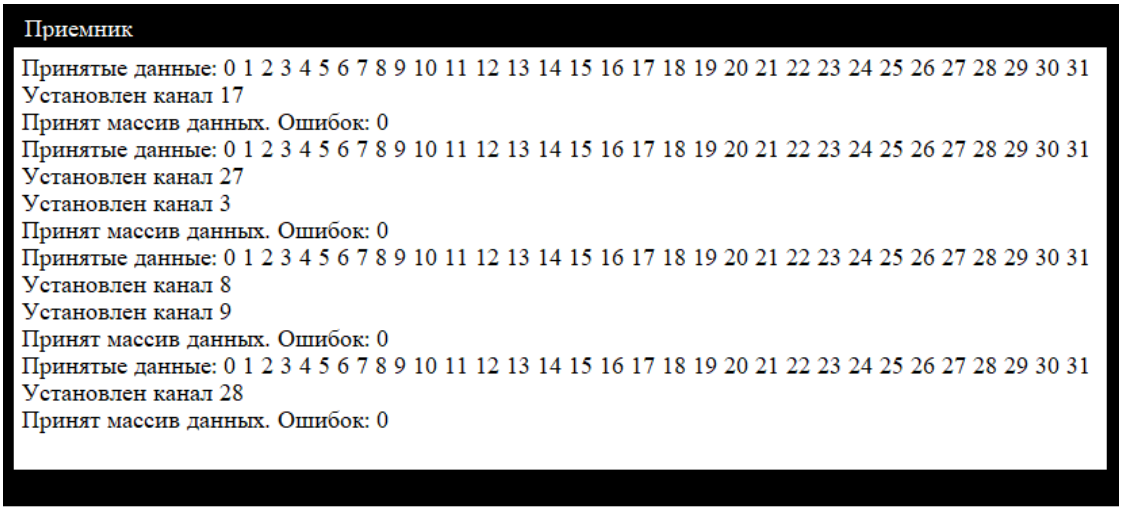

Рис. 3.25. Снимок экрана вывода информации с монитора порта
приемника во время работы

Так как используемые модули работают на частотах 2,4 ГГц, что является частотами стандарта Wi-Fi, возникают случаи, когда синхронизация переключения каналов сбивается по причине того, что в данный момент на этой частоте находится один из каналов Wi-Fi.

Это можно наблюдать на рис. 3.26, 3.27, где в процессе переключения каналов передатчику не удалось отправить сообщение, а приемнику – принять. На экране красным цветом будет выделен момент, когда транзакция не совершилась, зеленым цветом – когда все прошло успешно. После серии неудачных попыток модули переключаются на следующий канал в последовательности и совершают успешную транзакцию данных.

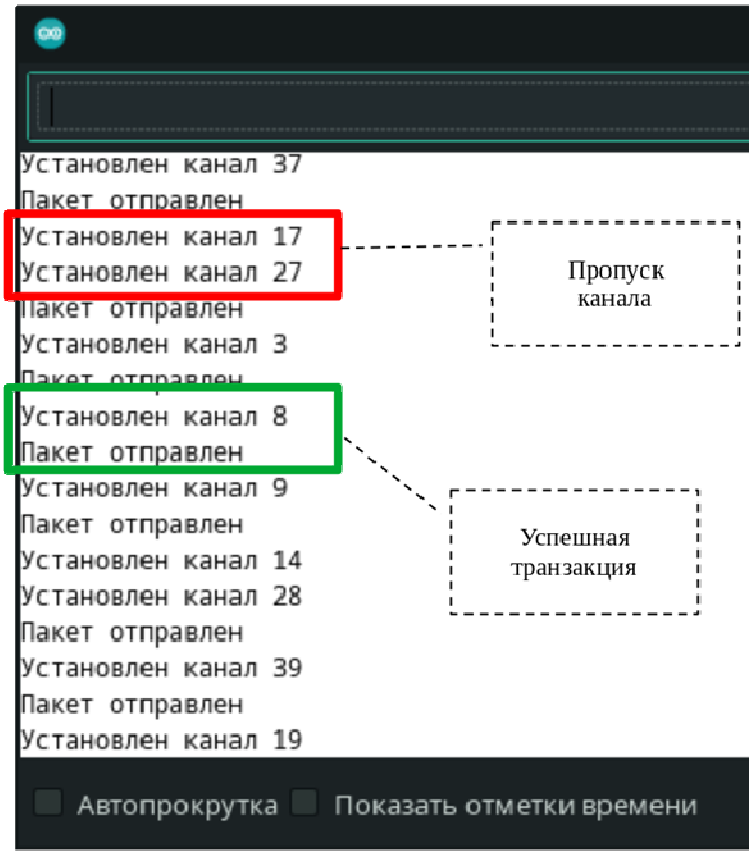

Рис. 3.26. Снимок экрана вывода информации с монитора порта передатчика при успешной и провальной транзакции

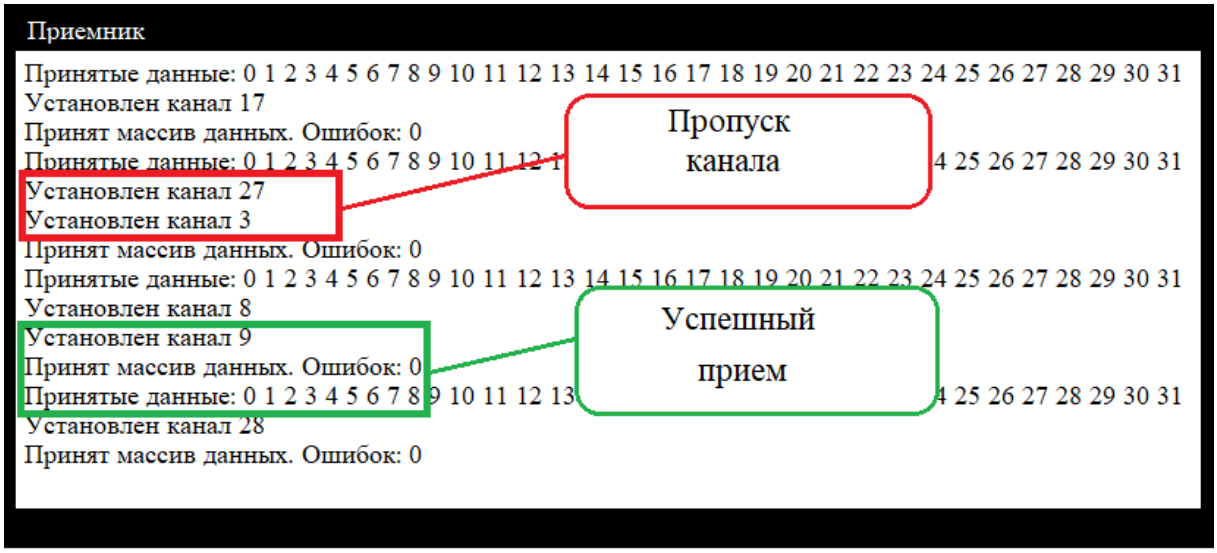

Рис. 3.27. Снимок экрана вывода информации с монитора порта приемника при успешной и провальной транзакции

В прил. 3 приведены тексты программ для приемника и передатчика с псевдослучайной перестройкой радиочастоты.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие известные варианты архитектуры микропроцессорных устройств вам известны?
2. В чем состоят особенности CISC и RISC архитектуры?
3. В чем состоит принцип работы аналого-цифровых и цифро-аналоговых преобразователей?
4. Какие основные интерфейсы современных микроконтроллеров вы можете назвать?
5. Перечислите основные характеристики современных микроконтроллеров.
6. Объясните работу циклов for и while языка программирования C++.
7. Назовите функции работы с параллельным интерфейсом платформы Arduino.
8. Каково предназначение функций setup и loop?
9. Приведите пример программы управления яркостью свечения светодиода на платформе Arduino.
10. Что представляет собой широтно-импульсная модуляция. В каких случаях она применяется?
11. Каковы основные характеристики современных трансиверов?
12. Перечислите основные функции библиотеки RF24 для программирования трансиверов.
13. Приведите пример программы для передачи сообщений с помощью трансивера.
14. Приведите пример программы для приема сообщений с помощью трансивера.
15. Что такое CRC?
16. В чем заключается помехоустойчивое кодирование?
17. Приведите пример программы кодера Хемминга.
18. В чем особенность программы декодера Хемминга 11/15? Приведите пример.
19. Каков принцип работы ППРЧ?
20. Приведите пример программы генератора псевдослучайной последовательности.

ЗАКЛЮЧЕНИЕ

В пособии приведен достаточно полный обзор современных микроконтроллеров, их архитектуры, устройств ввода/вывода и интегрированных аналого-цифровых и цифроаналоговых устройств; описаны основные интерфейсы UART/USART, SPI, JTAG, а также схемы их применения.

В книге изложены основы программирования микроконтроллеров на базе платформы Arduino как наиболее популярной на сегодняшний день и позволяющей быстро овладеть общими навыками программирования. Приведены описание платформы Arduino, цифровых и аналоговых входов и выходов, принципы организации программных прерываний по встроенному таймеру; дано представление об оперативной памяти и широтно-импульсной модуляции. Подробно указаны основные функции, типы данных и операторы программирования на языке C++.

Особое внимание уделено основам организации беспроводных систем связи: приведен обзор современных программируемых трансиверов и указаны наиболее важные характеристики беспроводных модулей передачи данных; описаны устройство и регистры программирования на примере беспроводного модуля NRF24L01; изложены принципы программирования приемника и передатчика для беспроводной системы связи. Раскрыто понятие помехоустойчивого кодирования на примере кодов Хемминга и Рида – Соломона. Рассмотрена система связи с псевдослучайной перестройкой радиочастоты.

Авторы надеются, что приведенный в пособии материал даст возможность студентам радиотехнических специальностей достаточно полно изучить принципы функционирования микроконтроллеров, правила их программирования, понять особенности формирования программных прерываний, определить принципы построения и программирования беспроводных систем связи и помехоустойчивой обработки информации.

В этих целях в приложениях приведены действующие программы, обеспечивающие помехоустойчивое кодирование и декодирование информации по кодам Хемминга и Рида – Соломона, а также реализация псевдослучайной перестройки радиочастоты. Авторы полагают, что приведенные программные продукты будут полезны при разработке и эксплуатации беспроводных систем передачи информации и других управляющих устройств на базе микроконтроллеров.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. История развития микроконтроллеров [Электронный ресурс]. – URL: https://volna.org/informatika/istoriya-razvitiya-mikrokontrollerov.html (дата обращения: 17.09.2022).
2. Микроконтроллеры от ведущих мировых производителей [Электронный ресурс]. – URL: http://www.mymcu.ru/ (дата обращения: 17.09.2022).
3. Интерфейсы микроконтроллеров [Электронный ресурс]. – URL: https://www.drive2.ru/b/2602560/ (дата обращения: 17.09.2022).
4. Современные микроконтроллеры. Архитектура, особенности различных типов микроконтроллеров [Электронный ресурс]. – URL: https://studfile.net/preview/7580551/page:6/ (дата обращения: 17.09.2022).
5. Микроконтроллеры. Краткий обзор [Электронный ресурс]. – URL: https://myrobot.ru/stepbystep/mc_meet.php (дата обращения: 17.09.2022).
6. Мир микроконтроллеров [Электронный ресурс]. – URL: https://microkontroller.ru/avr-microcontroller-projects/kak-ispolzovat-adc-v-avr-atmega16/ (дата обращения: 17.09.2022).
7. Преобразователи АЦП и ЦАП [Электронный ресурс]. – URL: https://siblec.ru/tekhnicheskie-nauki/tekhnicheskie-sredstva-avtomatizatsii-i-upravleniya/5-preobrazovateli-atsp-i-tsap-dac-adc (дата обращения: 17.09.2022).
8. Как работать с АЦП и ЦАП в микроконтроллерах [Электронный ресурс]. – URL: http://www.mymcu.ru/articles/kak-rabotat-atsp-tsap-mikrokontrollerah-silabs.html (дата обращения: 17.09.2022).
9. Википедия. JTAG [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/JTAG (дата обращения: 17.09.2022).
10. Handling external Interrupts with Arduino [Электронный ресурс]. – URL: https://gonium.net/md/2006/12/20/handling-external-interrupts-with-arduino/ (дата обращения: 17.09.2022).
11. AVR Libs Home Page [Электронный ресурс]. – URL: https://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html (дата обращения: 17.09.2022).
12. Программирование Arduino [Электронный ресурс]. – URL: https://arduino.ru/Reference/ (дата обращения: 17.09.2022).

13. Оппенгейм, А. Цифровая обработка сигналов / А. Оппенгейм. – 3-е изд., стер. – М. : Техносфера, 2012. – 1048 с. – ISBN 978-5-94836-329-5.

14. Шустов, М. А. Цифровая схемотехника. Основы построения / М. А. Шустов – М. : НиТ, 2018. – 320 с. – ISBN 978-5-94387-875-6.

15. Радиомодуль NRF24L01+ / PA+LNA 2.4G (Trema-модуль V2.0) [Электронный ресурс]. – URL: https://wiki.iarduino.ru/page/NRF24L01-trema/ (дата обращения: 24.04.2021).

16. Радиомодуль NRF24L01 [Электронный ресурс]. – URL: https://3d-diy.ru/wiki/arduino-moduli/radio-modul-nrf24l01/ (дата обращения: 16.01.2021).

17. Шмаков, С. Б. Энциклопедия радиолюбителя. Современная элементная база / С. Б. Шмаков. – 2-е изд., стер. – СПб. : Наука и техника, 2012. – 384 с. – ISBN 978-5-94387-859-6.

18. Голиков, А. М. Модуляция, кодирование и моделирование в телекоммуникационных системах. Теория и практика : учеб. пособие / А. М. Голиков. – Томск : Том. гос. ун-т систем упр. и радиоэлектроники, 2016. – 516 с. – (Учебная литература для вузов). – ISBN 978-5-8114-7828-6.

19. Баранов, В. Н. Применение микроконтроллеров AVR. Схемы, алгоритмы, программы / В. Н. Баранов. – 2-е изд., стер. – М. : Додэка XXI, 2006. – 288 с. – (Серия «Мировая электроника»). – ISBN 5-94120-075-7.

20. Белов, А. В. Микроконтроллеры AVR в радиолюбительской практике / А. В. Белов. – СПб. : Наука и техника, 2007. – 352 с. – ISBN 978-5-94387-365-2.

21. Ревич, Ю. В. Занимательная электроника / Ю. В. Ревич. – 3-е изд., перераб. и доп. – СПб. : БХВ-Петербург, 2015. – 576 с. – ISBN 978-5-9775-6701-5.

22. Подключение модуля nRF24L01+ к Arduino – соединяем две arduino по радиоканалу [Электронный ресурс]. – URL: https://micro-pi.ru/подключение-модуля-nrf24l01-к-arduino// (дата обращения: 04.10.2020).

23. Левашов, Ю. А. Прием и обработка сигналов : учеб. пособие / Ю. А. Левашов. – Владивосток : Изд-во ВГУЭС, 2004. – 108 с. – ISBN 978-5-7996-1497-3.

24. Подключение модулей связи 2,4 ГГц на базе чипов nRF24L01+ к микроконтроллеру [Электронный ресурс]. – URL: https://aterlux.ru/article/nrf24l01p#__h3_38 (дата обращения: 11.05.2021).

25. Колосовский, Е. А. Устройства приема и обработки сигналов / Е. А. Колосовский. – 2-е изд., стер. – М. : Горячая линия – Телеком, 2012. – 457 с. – ISBN 978-5-9912-0265-7.

26. Коберниченко, В. Г. Основы цифровой обработки сигналов : учеб. пособие / В. Г. Коберниченко. – Екатеринбург : Изд-во Урал. ун-та, 2018. – 150 с. – ISBN 978-5-7996-2464-4.

27. Low-Power RF Devices and the ISM Bands [Электронный ресурс]. – URL: https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/electromagnetic-spectrum/low-power-rf-devices-and-the-ism-bands/# (дата обращения: 14.03.2021).

28. How nRF24L01+Wireless Module Works & Interface with Arduino [Электронный ресурс]. – URL: https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/ (дата обращения: 26.02.2021).

29. Тяпичев, Г. Как построить трансивер (Азбука УКВ) / Г. Тяпичев. – М. : ДМК-пресс, 2005. – 432 с. – ISBN 978-5-9706-0001-6.

30. Владимиров, С. С. Математические основы теории помехоустойчивого кодирования : учеб. пособие / С. С. Владимиров. – СПб. : СПбГУТ, 2016. – 96 с. – ISBN 978-5-89160-131-4.

31. Кудряшов, Б. Д. Основы теории кодирования : учеб. пособие / Б. Д. Кудряшов. – СПб. : БХВ-Петербург, 2016. – 400 с. – (Учебная литература для вузов). – ISBN 978-5-9775-3527-4.

32. Радиопередающие устройства в системах радиосвязи : учеб. пособие / Ю. Т. Зырянов [и др.]. – 4-е изд., стер. – СПб. : Лань, 2020. – 176 с. – ISBN 978-5-8114-5532-4.

33. Радиоприемные устройства в системах радиосвязи : учеб. пособие / Ю. Т. Зырянов [и др.]. – 2-е изд., стер. – СПб. : Лань, 2018. – 320 с. – ISBN 978-5-8114-2589-1.

34. Каналы связи [Электронный ресурс]. – URL: http://www.e-biblio.ru/book/bib/01_informatika/infteh/book/docs/piece143.htm (дата обращения: 27.02.2021).

35. Виды связи общего назначения [Электронный ресурс]. – URL: http://www.e-biblio.ru/book/bib/01_informatika/infteh/book/docs/piece142.htm (дата обращения: 27.02.2021).

36. Евстифеев, А. В. Микроконтроллеры AVR семейства Classic фирмы ATMEL : учеб. пособие / А. В. Евстифеев. – 6-е изд., стер. – М. : Додэка XXI, 2010. – 285 с. – ISBN 978-5-94120-219-5.

37. Магда, Ю. С. Современные микроконтроллеры. Архитектура, программирование, разработка устройств / Ю. С. Магда. – М. : ДМК Пресс, 2017. – 224 с. – ISBN 978-5-97060-551-6.

38. Плаксиенко, В. С. Радиоприемные устройства и телевидение : учеб. пособие / В. С. Плаксиенко, Н. Е. Плаксиенко. – Ростов н/Д. : ЮФУ, 2018. – 99 с. – ISBN 978-5-9275-2955-1.

39. Никитин, Н. П. Прием и обработка сигналов в цифровых системах передачи : учеб. пособие / Н. П. Никитин, В. И. Лузин. – Екатеринбург : УрФУ, 2013. – 124 с. – ISBN 978-5-7996-1022-7.

40. Фриск, В. В. Теория электрических цепей, схемотехника телекоммуникационных устройств, радиоприемные устройства систем мобильной связи, радиоприемные устройства систем радиосвязи и радиодоступа : учеб. пособие / В. В. Фриск, В. В. Логвинов. – М. : СОЛОН-Пресс, 2016. – 480 с. – ISBN 978-5-91359-167-8.

ПРИЛОЖЕНИЯ

Приложение 1

Кодек Хемминга

Программа для передатчика с кодом Хемминга

```
#include <SPI.h>
#include <RF24.h>
RF24 radio(10, 9); // порты D9, D10: CSN CE
const uint32_t pipe = 111156789; // адрес рабочей трубы;

byte data[32];
byte Info[32];
byte d[8];
void setup() {
  Serial.begin(9600);
  Serial.println("Transmitter-Encoder Hemming ON");

  radio.begin();   // инициализация
  delay(2000);
  radio.setDataRate(RF24_1MBPS);     // скорость обмена данными
                                        RF24_1MBPS или RF24_2MBPS
  radio.setCRCLength(RF24_CRC_8); // размер контрольной суммы
                                        8 bit или 16 bit
  radio.setPALevel(RF24_PA_HIGH);  // уровень питания усилителя
                                        RF24_PA_MIN, RF24_PA_LOW,
                                    RF24_PA_HIGH and RF24_PA_MAX
  radio.setChannel(0x6f);    // установка канала
  radio.setAutoAck(false);  // автоответ
  radio.powerUp();           // включение или пониженное потребление
                                    powerDown – powerUp
  radio.stopListening();    // радиоэфир не слушаем, только передача
  radio.openWritingPipe(pipe);    // открыть трубу на отправку
```

```
 Serial.println("Info:");
  for(int i=0;i<32;i++) {
   Info[i]=rand()%16;
   Serial.print(String(Info[i])+" ");
   }
   Serial.println(" ");
   Serial.println("Encode:");

   for(int k=0;k<32;k++)
 {
  byte c=Info[k];
  d[7]=c&1;
  d[6]=(c>>1)&1;
  d[5]=(c>>2)&1;
  d[3]=(c>>3)&1;
  d[1]=d[3]^d[5]^d[7];
  d[2]=d[3]^d[6]^d[7];
  d[4]=d[5]^d[6]^d[7];
  d[0]=0;
  data[k]=0;
  for(int i=0;i<8;i++)
  {
   data[k]=data[k]|(d[i]<<i);
   }
   Serial.print(String(data[k]>>1)+" ");
 }
}

void loop() {
  radio.write(&data[0], 32);
}
```

Программа для приемника с кодом Хемминга

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#define BLOCKS 10000
#define POINT 100
RF24 radio(10, 9); // порты D9, D10: CSN CE

const uint32_t pipe = 111156789; // адрес рабочей трубы;
byte data[32];
byte Info[32];
float err,errk;
int scn; //счетчик циклов прослушивания эфира
int sg; //счетчик числа принятых пакетов с передатчика

void setup() {
  Serial.begin(9600);
  Serial.println("Receiver-Decoder Hemming ON");

  radio.begin();    // инициализация
  delay(2000);
  radio.setDataRate(RF24_1MBPS);   // скорость обмена данными
                                        RF24_1MBPS или RF24_2MBPS
  radio.setCRCLength(RF24_CRC_8);   // размер контрольной суммы
                                          8 bit или 16 bit
  radio.disableCRC();
  radio.setChannel(0x6f);   // установка канала
  radio.setAutoAck(false);   // автоответ
  radio.openReadingPipe(1, pipe);   // открыть трубу на прием
  radio.startListening();   // прием
  err=0;
  errk=0;
  sg=0;
  for(int i=0;i<32;i++) Info[i]=rand()%16;
}
```

```
void loop() {
  if (sg < BLOCKS)
  {   // прослушивание эфира
    if (radio.available())
   {

     radio.read(data, 32);
     if(sg%POINT==0) Serial.print(".");
     for(int m=0;m<32;m++)
    {
       byte c=data[m];
       byte c1,c2,c4;

       c1=((c>>1)&1)^((c>>7)&1)^((c>>5)&1)^((c>>3)&1);
       c2=((c>>2)&1)^((c>>7)&1)^((c>>6)&1)^((c>>3)&1);
       c4=((c>>4)&1)^((c>>7)&1)^((c>>6)&1)^((c>>5)&1);
       byte e=(c4<<2)|(c2<<1)|c1;
       c=c^(1<<e);
       byte cc=((c>>7)&1)|((c>>5)&2)|((c>>3)&4)|(c&8);
       cc=cc^Info[m];
      for(int k=0;k<4;k++)
       {
         errk+=cc&1;
         cc=cc>>1;
        }

         c=data[m];
         cc=((c>>7)&1)|((c>>5)&2)|((c>>3)&4)|(c&8);
         cc=cc^Info[m];
        for(int k=0;k<4;k++)
       {
        err+=cc&1;
        cc=cc>>1;
        }
    }
    sg++;
  }
```

```
 } else
 {   // всего принято
  Serial.println();
   Serial.print("Принято: " + String(sg) + " блоков; BER=");
   Serial.print(err/sg/32.0/4.0,6);
   Serial.print("; BER_K=");
   Serial.println(errk/sg/32.0/4.0,6);
   sg = 0;
   err=0;
   errk=0;
   scn = 0;

 }
  scn++;
// delay(20);
  if (sg >= BLOCKS) {sg = BLOCKS;}     // защита от переполнения счет-
                                          чика
}
```

Приложение 2

Кодек Рида – Соломона

Программа для передатчика с кодом Рида – Соломона

```
#include <SPI.h>
#include <RF24.h>
RF24 radio(10, 9); // порты D9, D10: CSN CE
const uint32_t pipe = 111156789; // адрес рабочей трубы;

#define mMax 8
#define nMax 31
#define tMax 31

int m;
int k;
int n;
int N;
int Max;
byte DATA[nMax];
int pol[mMax] = {3,7,11,19,37,67,137,285};
int alpha_to[nMax + 1];
int index_of[nMax + 1];
int V;
int G[2*tMax+1];
byte TRANSMITED[nMax];
int g1[2*tMax+1];

void Init_Koder()
{
  int p[mMax+1];
  int i,s,j;
  s=pol[m-1];
  for (i=0; i<=m;i++)
  {
    if (s%2==1) {s=(s-1)/2; p[i]=1;}
    else {s=s/2; p[i]=0;}
  }
```

```
// Galua
    int mask;
    mask = 1; alpha_to[m] = 0;
    for (i = 0; i < m; i++)
     {
      alpha_to[i] = mask;
      index_of[alpha_to[i]] = i;
      if (p[i] != 0) alpha_to[m] ^= mask;
      mask <<= 1;
     }
   index_of[alpha_to[m]] = m; mask >>= 1;
   for (i = m + 1; i < N; i++)
    {
      if (alpha_to[i – 1] >= mask)
      alpha_to[i] = alpha_to[m] ^ ((alpha_to[i – 1] ^ mask) << 1);
     else
      alpha_to[i] = alpha_to[i – 1] << 1;
  index_of[alpha_to[i]] = i;
     } index_of[0] = -1;

// Generiruuschiy polinom
int X0[2*tMax+1];
int alpha=2;
int kd;
   g1[0]=3;
   g1[1]=index_of[alpha_to[1]^alpha_to[2]];
   g1[2]=0;
   for (i = 3; i < tMax+1; i++) {

g1[i]=0;
  }

   kd=n-k-2;
   for (i=kd; i>0; i--)
  {
     for (j=0; j<=alpha; j++)
```

```
  {
    X0[j]=g1[j];
  }
    for (j=1; j<=alpha; j++)
  {
    g1[j]=X0[j-1];
  }
 alpha++;

 g1[0]=(X0[0]+0+alpha)%N;
 for (j=1; j<alpha; j++)
 {
 g1[j]=index_of[alpha_to[g1[j]]^alpha_to[(X0[j]+0+alpha)%N]];
 }
 }
}
//------------------

void Koder()
{

int i,j;
     int feedback;

 for (i = 0; i < n – k; i++) TRANSMITED[i] = 0;
   for (i = k – 1; i >= 0; i--)
    {
      feedback = index_of[DATA[i] ^ TRANSMITED[n – k – 1]];
      if (feedback != -1)
     {
     for (j = n – k – 1; j > 0; j--)
      if (g1[j] != -1) TRANSMITED[j] = TRANSMITED[j – 1] ^
      alpha_to[(g1[j] + feedback) % N];
      else
       TRANSMITED[j] = TRANSMITED[j – 1];
       TRANSMITED[0] = alpha_to[(g1[0] + feedback) % N];
     }
```

```
  else
  {
   for (j = n – k – 1; j > 0; j--) TRANSMITED[j] = TRANSMITED[j – 1];
   TRANSMITED[0] = 0;
   }
 }

 for (i=n-k; i<n; i++)
  {
   TRANSMITED[i]=DATA[i-n+k];
  }
}
//-----------------

void setup() {
  Serial.begin(9600);
  Serial.println("Transmitter-Encoder Reed-Solomon ON");

  radio.begin();   // инициализация
  delay(2000);
  radio.setDataRate(RF24_1MBPS);  // скорость обмена данными
                                              RF24_1MBPS или RF24_2MBPS
  radio.setCRCLength(RF24_CRC_8);   // размер контрольной суммы
                                                 8 bit или 16 bit
  radio.setPALevel(RF24_PA_LOW); // уровень питания усилителя
                                                RF24_PA_MIN, RF24_PA_LOW,
                                           RF24_PA_HIGH and RF24_PA_MAX
  radio.setChannel(0x6f);   // установка канала
  radio.setAutoAck(false);   // автоответ
  radio.powerUp();   // включение или пониженное потребление
                              powerDown – powerUp
  radio.stopListening();   // радиоэфир не слушаем, только передача
  radio.openWritingPipe(pipe);   // открыть трубу на отправку

m=4;
k=8;
n=14;
```

```
N=(1<<m)-1;
Max=N+1;
  for(int i=0;i<k;i++) DATA[i]=rand()%Max;
  Init_Koder();
  Koder();
  Serial.println("m="+String(m)+", k="+String(k)+", n="+String(n));
  for(int i=n-1;i>=0;i--)
  {if(i==n-k-1)Serial.print(" ");
  Serial.print(String(TRANSMITED[i])+" ");
  }
}

void loop() {

 radio.write(&TRANSMITED[0], n);
}
```

Программа для приемника с кодом Рида – Соломона

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

RF24 radio(10, 9); // порты D9, D10: CSN CE

const uint32_t pipe = 111156789;   // адрес рабочей трубы;
#define BLOCKS 10000
#define mMax 8
#define nMax 31
#define tMax 31
#define P 2
#define NORM 100
int m;
int k;
int n;
int N;
int L;
```

```
int Chen;
int count;
int Max;

int pol[mMax] = {3, 7, 11, 19, 37, 67, 137, 285};
int alpha_to[nMax + 1];
int index_of[nMax + 1];

int err0;
int err_k,err_kk;
int err_n;
int ch;
int zn;
int V;
int G[2 * tMax + 1];
byte Info[32];
byte TRANSMITED[32];
int Xp[2 * tMax];
int X[2 * tMax + 2];
int s[2 * tMax + 1];

float err, errk;
int scn;   // счетчик циклов прослушивания эфира
int sg;    // счетчик числа принятых пакетов с передатчика

void Init_Koder()
{
  int p[mMax + 1];
  int g1[2 * tMax + 1];
  int i, s, j;
  s = pol[m – 1];
  for (i = 0; i <= m; i++)
  {
    if (s % 2 == 1) {
      s = (s – 1) / 2;
      p[i] = 1;
    }
```

```
  else {
    s = s / 2;
    p[i] = 0;
  }
 }

// Galua
 int mask;
 mask = 1; alpha_to[m] = 0;
 for (i = 0; i < m; i++)
 {
  alpha_to[i] = mask;
  index_of[alpha_to[i]] = i;
  if (p[i] != 0) alpha_to[m] ^= mask;
  mask <<= 1;
 }
 index_of[alpha_to[m]] = m; mask >>= 1;
 for (i = m + 1; i < N; i++)
 {
  if (alpha_to[i – 1] >= mask)
   alpha_to[i] = alpha_to[m] ^ ((alpha_to[i – 1] ^ mask) << 1);
  else
   alpha_to[i] = alpha_to[i – 1] << 1;
   index_of[alpha_to[i]] = i;
 } index_of[0] = -1;
// Generiruuschiy polinom
 int X0[2 * tMax + 1];
 int alpha = 2;
 int kd;
 g1[0] = 3;
 g1[1] = index_of[alpha_to[1] ^ alpha_to[2]];
 g1[2] = 0;
 for (i = 3; i < tMax + 1; i++) {

  g1[i] = 0;
 }
 kd = n – k – 2;
 for (i = kd; i > 0; i--)
```

```
 {
  for (j = 0; j <= alpha; j++)
  {
   X0[j] = g1[j];
  }
  for (j = 1; j <= alpha; j++)
  {
   g1[j] = X0[j – 1];
  }
  alpha++;

 g1[0] = (X0[0] + 0 + alpha) % N;
 for (j = 1; j < alpha; j++)
  {
 g1[j] = index_of[alpha_to[g1[j]] ^ alpha_to[(X0[j] + 0 + alpha) % N]];
  }
 }
}

int Dekoder()
{
 int i, j, q;
 // int s[2*tMax + 1];

 int syn_error = 0, root[tMax], loc[tMax], z[2 * tMax + 1], reg[tMax + 1];
 // Vycheslenie sindromov
for (i = 0; i < n; i++) TRANSMITED[i] = index_of[TRANSMITED[i] & N];
 s[0] = -1;
 for (i = 1; i <= n – k; i++)
 {
 s[i] = 0;
 for (j = 0; j < n; j++)
 if (TRANSMITED[j] != 255) s[i] ^= alpha_to[(TRANSMITED[j] + i * j)
% N];
 if (s[i] != 0) syn_error = 1;
 s[i] = index_of[s[i]];
 }
 if (syn_error)
 {
```

```
  //-----------polinom stiraniy-------------------------
  for (i = 0; i < 2 * tMax + 1; i++) G[i] = 0;

  //--------------------Algoritm Berlikampa-Messi-------------
  int B[2 * tMax + 2];
  int M[2 * tMax + 2];
  int r, dr, bl, ml, xl;
  r = V;
  L = V;
  bl = V;
  xl = V;
  for (i = 0; i < n – k + 2; i++) {
    X[i] = 0;
    B[i] = -1;
    M[i] = 0;
   }
  for (i = 0; i <= V; i++) {
    X[i] = G[i];
    B[i] = G[i];
   }
  while (r != n – k)
 {
r++;

if (s[r] != -1) dr = alpha_to[s[r]];
else dr = 0;
for (j = 1; j <= L; j++)
  if ((X[j] != -1) && (s[r – j] != -1)) dr ^= alpha_to[(X[j] + s[r – j]) % N];

dr = index_of[dr];
if (dr == -1)
{
  for (j = bl; j >= 0; j--) B[j + 1] = B[j];
  bl++; B[0] = -1;
}
else
{
```

```
M[0] = X[0];
for (j = 1; j <= bl + 1; j++)
{
  int aa = 0;
  int bb = 0;
  if (B[j – 1] != -1) aa = alpha_to[(dr + B[j – 1]) % N];
  if (X[j] != -1) bb = alpha_to[X[j]];
  if (j > L) M[j] = aa;
  else M[j] = aa ^ bb;
  M[j] = index_of[M[j]];
}
ml = bl;

if (2 * L > V + r – 1)
{
  for (j = 0; j <= bl + 1; j++) {
    X[j] = M[j];
    xl = bl + 1;
  }
  for (j = bl; j >= 0; j--) B[j + 1] = B[j];
  bl++; B[0] = -1;
  }
 else
{
  bl = xl;
  for (j = 0; j <= xl; j++)
  {
    if (X[j] != -1) B[j] = (X[j] + N – dr) % N;
    else B[j] = -1;
  }
  L = r + V – L; xl = ml + 1;
  for (j = 0; j <= xl; j++) X[j] = M[j];
  }
 }
}
```

```
//----------------------Algoritm Chena---------------------------------
count = 0;
if (L – V <= (n – k – V) / 2)
{
  for (i = 0; i <= L; i++) reg[i] = X[i];
  for (i = 1; i <= N; i++)
  {
    q = 1 ;
    for (j = 1; j <= L; j++)

     if (reg[j] != -1)
    {
      reg[j] = (reg[j] + j) % N;
      q ^= alpha_to[reg[j]];
    }
   if (!q)
  {
    root[count] = i;
    loc[count] = N – i;
    if (i < N – n + 1) break;
    count++;
   }
  }
}
else
{
for (i = 0; i < n; i++) if (TRANSMITED[i] != 255) TRANSMITED[i] =
alpha_to[TRANSMITED[i]]; else TRANSMITED[i] = 0 ;
// LL=10;
   return 3;
}
```

```
//----------------------------Algoritm Forni-----------------------

if (count == L)
{

    for (j = 1; j <= L; j++)
   { if (X[j] != -1)
       if (j % 2 == 1) Xp[j – 1] = X[j];
       else Xp[j – 1] = -1;
    else Xp[j – 1] = -1;
}

for (int jj = 0; jj < L; jj++) z[jj] = 0;

for (i = 0; i < L; i++)
 for (j = 0; j < L – i; j++)
  if ((s[i + 1] != -1) && (X[j] != -1)) z[i + j] ^= alpha_to[(s[i + 1] + X[j]) % N];


 for (j = 0; j < L; j++) {
   z[j] = index_of[z[j]];
 }

 for (i = 0; i < n; i++) if (TRANSMITED[i] != 255) TRANSMITED[i] =
alpha_to[TRANSMITED[i]]; else TRANSMITED[i] = 0;

 for (i = 0; i < L; i++)
 {
   ch = 0;
   zn = 0;
   err0 = 0;
   for (j = 0; j < L; j++)
   {

   if (z[j] != -1) ch ^= alpha_to[(z[j] + j * root[i]) % N];
   if (Xp[j] != -1) zn ^= alpha_to[(Xp[j] + j * root[i]) % N];
   }
  if ((ch != 0) && (zn != 0))
```

```
  {
    err0 = (index_of[ch] – index_of[zn] + N) % N;
    Chen = alpha_to[err0];
    TRANSMITED[loc[i]] ^= alpha_to[err0];

  }
  // else err0=-1;
 }
  return 1;
}

 else

 {
    for (i = 0; i < n; i++) if (TRANSMITED[i] != 255) TRANSMITED[i] =
alpha_to[TRANSMITED[i]]; else TRANSMITED[i] = 0;
    return 2;
   }
 }
 else
    for (i = 0; i < n; i++) if (TRANSMITED[i] != 255) TRANSMITED[i] =
alpha_to[TRANSMITED[i]]; else TRANSMITED[i] = 0;
    return 0;
}

void setup() {
  Serial.begin(9600);
  Serial.println("Receiver-Decoder Reed-Solomon ON");

  radio.begin();   // инициализация
  delay(2000);
  radio.setDataRate(RF24_1MBPS);   // скорость обмена данными
RF24_1MBPS или RF24_2MBPS
  radio.setCRCLength(RF24_CRC_8);   // размер контрольной суммы
                                          8 bit или 16 bit
  radio.disableCRC();
  radio.setChannel(0x6f);    // установка канала
```

```
  radio.setAutoAck(false);   // автоответ
  radio.openReadingPipe(1, pipe);   // открыть трубу на прием
  radio.startListening();   // прием
  err = 0;
  errk = 0;
  err_k = 0;
  err_kk = 0;
  err_n = 0;
  m = 4;
  k = 8;
  n = 14;
  N = (1 << m) – 1;
  Max = N + 1;
  V = 0;
  Init_Koder();
  for (int i = 0; i < k; i++)
 { Info[i] = rand() % Max;
 }

sg=0;
  Serial.println("m=" + String(m) + ", k=" + String(k) + ", n=" + String(n));
}

void loop() {
  if (sg < BLOCKS)
  {   // прослушивание эфира
    if (radio.available())
   {
     radio.read(TRANSMITED, n);
if(sg%NORM==0) Serial.print(".");
     for (int qqq = n – 1; qqq >= n – k; qqq--)
    {

     byte c = TRANSMITED[qqq];

     c = c ^ Info[qqq – (n – k)];
     for (int qqq1 = 0; qqq1 < m; qqq1++)
```

```
   {
    err += c & 1;
    c = c >> 1;
   }
  }

 for (int qqq = n – 1; qqq >= n – k; qqq--)
 {
  if ((TRANSMITED[qqq]&N) != Info[qqq – (n – k)])
  {
   err_k++;
  }
 }
 Dekoder();

 for (int qqq = n – 1; qqq >= n – k; qqq--)
{
  byte c = TRANSMITED[qqq]&N;

  c = c ^ Info[qqq – (n – k)];

  if(c!=0) err_kk++;
  for (int qqq1 = 0; qqq1 < m; qqq1++)
  {
   errk += c & 1;
   c = c >> 1;
  }
 }
sg++;

 }
} else {   // всего принято
{
  Serial.println();
  Serial.print("Принято: " + String(sg) + " блоков; BER=");
  Serial.print(err / sg / k / m, 6);
  Serial.print("; BER_K=");
```

```
  Serial.print(errk / sg / k / m, 6);
   Serial.print("; S_ERR=");
   Serial.print(err_k);
   Serial.print("; S_ERR_K=");
   Serial.print(err_kk);
   Serial.println();
   sg = 0;
   err = 0;
   errk = 0;
   err_k = 0;
   err_kk = 0;
   }

 }
// scn++;
 // delay(20);

 if (scn >= BLOCKS) {
   sg = BLOCKS; //защита от переполнения счетчика
   }
}
```

Приложение 3

Псевдослучайная перестройка радиочастоты

Программа передатчика с ППРЧ

```
#include <SPI.h>
#include "RF24.h"
#include "printf.h"
#include <Arduino.h>

/** Настройки пользователя */
#define CHANNELS_CNT 10
#define CHANNEL_START 0
#define CHANNELS_STOP 40
#define TRANSMIT_TRYING 10
#define RECEIVE_TRYING 30
#define RANDOM_INIT_NUM 17
#define OUT_POWER  RF24_PA_MIN


const uint64_t pipes[2] = { 0xABCDABCD71LL, 0x544d52687CLL };

#define SYNC_NUM 0x55
#define DATA_SIZE 32
uint8_t data_to_send[DATA_SIZE] = {0};
uint8_t channels[CHANNELS_CNT] = {0};

RF24 radio(10,9);

static void init_channels(int init_num)
{
  randomSeed(init_num);
  printf("Номера каналов: ");
  for (int i = 0; i < CHANNELS_CNT; ++i)
```

```
   {
    channels[i] = ((random(CHANNEL_START, CHANNELS_STOP)))%125;
    printf("%d ", channels[i]);
   }
   printf("\n");
}

static void init_send_arr(uint8_t *arr)
{
  for (int i = 0; i < DATA_SIZE; ++i)
  {
   data_to_send[i] = i%256;
   }
}

static void change_channel(void)
{
  static uint8_t chan = 0;

  radio.setChannel(channels[chan]);
  printf("Установлен канал %d\n", channels[chan]);
  chan = (chan+1)%CHANNELS_CNT;
}

static FILE uartout = { 0 };
static int uart_putchar (char c, FILE *stream)
{
  if( c == '\n' )
    Serial.write('\r');
  Serial.write(c) ;
  return 0 ;
}
```

```
void setup(void)
{
 fdev_setup_stream(&uartout, uart_putchar, NULL, _FDEV_SETUP_WRITE);
 stdout = &uartout;
 Serial.begin(9600);

 radio.begin();
 radio.setPALevel(OUT_POWER);
 radio.setDataRate(RF24_1MBPS);
 radio.disableCRC();
 radio.setRetries(0,15);
 radio.setAutoAck(1);

 radio.openWritingPipe(pipes[0]);
 radio.openReadingPipe(1,pipes[1]);

 init_channels(RANDOM_INIT_NUM);
 init_send_arr(data_to_send);

 printf("Запуск программы\n");
 change_channel();
}

int transmit_cnt = 0;

void loop()
{
   if (radio.write(data_to_send, DATA_SIZE))
  {
   printf("Пакет отправлен\n");
   change_channel();
  }
  else
  {
```

```
   transmit_cnt++;
   if (transmit_cnt >= TRANSMIT_TRYING)
  {
   change_channel();
   transmit_cnt = 0;
  }
 }
 delay(50);
}
```

Программа приемника с ППРЧ

```
#include <SPI.h>
#include "RF24.h"
#include "printf.h"
#include <Arduino.h>

/** Настройки пользователя */
#define CHANNELS_CNT 10
#define CHANNEL_START 0
#define CHANNELS_STOP 40
#define TRANSMIT_TRYING 10
#define RECEIVE_TRYING 30
#define RANDOM_INIT_NUM 17
#define OUT_POWER  RF24_PA_MIN

const uint64_t pipes[2] = { 0xABCDABCD71LL, 0x544d52687CLL };

#define DATA_SIZE 32
uint8_t data_to_recv[DATA_SIZE] = {0};
uint8_t channels[CHANNELS_CNT] = {0};

RF24 radio(10,9); //Nano
//RF24 radio(9,8); //Uno
```

```
static void init_channels(int init_num)
{
  randomSeed(init_num);
  printf("Номера каналов: ");
  for (int i = 0; i < CHANNELS_CNT; ++i)
  {
   channels[i] = ((random(CHANNEL_START, CHANNELS_STOP)))%125;
   printf("%d ", channels[i]);
  }
  printf("\n");
}

static void change_channel(void)
{
   static uint8_t chan = 0;

   radio.setChannel(channels[chan]);
   printf("Установлен канал %d\n", channels[chan]);
   chan = (chan+1)%CHANNELS_CNT;
}

static uint32_t check_data(uint8_t *data)
{
   uint32_t err_cnt = 0;
   for (int i = 0; i < DATA_SIZE; ++i)
   {
   if (data[i] != (i%256))
    err_cnt+=1;
  }
  return err_cnt;
}
```

```
static FILE uartout = { 0 };
static int uart_putchar (char c, FILE *stream)
{
  if( c == '\n' )
    Serial.write('\r');
  Serial.write(c) ;
  return 0 ;
}

void setup(void)
{
 fdev_setup_stream(&uartout, uart_putchar, NULL, _FDEV_SETUP_WRITE);
 stdout = &uartout;
 Serial.begin(9600);

 radio.begin();
 radio.setPALevel(RF24_PA_MAX);
 radio.setDataRate(RF24_1MBPS);
 radio.disableCRC();
 radio.setAutoAck(1);

 radio.openWritingPipe(pipes[1]);
 radio.openReadingPipe(1,pipes[0]);

 init_channels(RANDOM_INIT_NUM);
 radio.startListening();
 printf("Запуск программы\n");
 change_channel();
}

int trying = 0;
```

```
void loop()
{
  trying = 0;
  do {
    trying++;
    delay(2);
  } while (!radio.available() && trying != RECEIVE_TRYING);

  if (radio.available())
  {
    radio.read(data_to_recv, DATA_SIZE);

    uint32_t err_cnt = check_data(data_to_recv);
    printf("Принят массив данных. Ошибок: %lu\n", err_cnt);
    printf("Принятые данные: ");
    for (int i = 0; i < DATA_SIZE; ++i)
      printf("%d ", data_to_recv[i]);
    printf("\n");
    memset(data_to_recv, 0, DATA_SIZE);
   }
   change_channel();
   delay(50);
}
```

Учебное издание

САМОЙЛОВ Сергей Александрович
САМОЙЛОВ Владислав Сергеевич

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ
ДЛЯ БЕСПРОВОДНЫХ СИСТЕМ СВЯЗИ

Учебно-практическое пособие

Редактор А. П. Володина

Технические редакторы Ш. Ш. Амирсейидов, Н. В. Пустовойтова

Компьютерная верстка Л. В. Макаровой

Выпускающий редактор А. А. Амирсейидова

Подписано в печать 31.03.23.
Формат 60×84/16. Усл. печ. л. 9,53. Тираж 30 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.